UNIVERSIDADE FEDERAL FLUMINENSE

AUGUSTO PIZANO VIEIRA BELTRÃO

HEURÍSTICA HÍBRIDA APLICADA AO PROBLEMA DO CAIXEIRO VIAJANTE COM SELEÇÃO DE HOTÉIS

NITERÓI

2020

UNIVERSIDADE FEDERAL FLUMINENSE

AUGUSTO PIZANO VIEIRA BELTRÃO

HEURÍSTICA HÍBRIDA APLICADA AO PROBLEMA DO CAIXEIRO VIAJANTE COM SELEÇÃO DE HOTÉIS

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense, como requisito parcial para obtenção do Grau de Mestre em Computação. Área de Concentração: Algoritmos e Otimização.

Orientador:
LUIZ SATORU OCHI

Co-orientador:
JOSÉ ANDRÉ DE MOURA BRITO

NITERÓI
2020

Ficha catalográfica automática - SDC/BEE
Gerada com informações fornecidas pelo autor

B453h Beltrão, Augusto Pizano Vieira
Heurística Híbrida Aplicada ao Problema do Caixeiro Viajante com Seleção de Hotéis / Augusto Pizano Vieira Beltrão ; Luiz Satoru Ochi, orientador ; José André de Moura Brito, coorientador. Niterói, 2020.
75 f.

Dissertação (mestrado)-Universidade Federal Fluminense, Niterói, 2020.

DOI: http://dx.doi.org/10.22409/PGC.2020.m.14783357773

1. Problema do Caixeiro Viajante Com Seleção de Hotéis. 2. Planejamento de Rotas. 3. Metaheurísticas. 4. Produção intelectual. I. Ochi, Luiz Satoru, orientador. II. Brito, José André de Moura, coorientador. III. Universidade Federal Fluminense. Instituto de Computação. IV. Título.

CDD -

Bibliotecário responsável: Sandra Lopes Coelho - CRB7/3389

AUGUSTO PIZANO VIEIRA BELTRÃO

HEURÍSTICA HÍBRIDA APLICADA AO PROBLEMA DO CAIXEIRO VIAJANTE COM SELEÇÃO DE HOTÉIS

Dissertação apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense, como requisito parcial para obtenção do Grau de Mestre em Computação. Área de Concentração: Algoritmos e Otimização.

Aprovada em Março de 2020.

BANCA EXAMINADORA

Prof. Luiz Satoru Ochi – Orientador, UFF

Prof. José André de M. Brito – Co-orientador, ENCE/IBGE

Prof. Nelson Maculan Filho, UFRJ

Prof. Fábio Protti, UFF

Prof. Gustavo Silva Semaan, UFF

NITERÓI

2020

A Deus. A meus pais.

AGRADECIMENTOS

Agradeço a Deus, por me dar a vida e todas as demais bençãos recebidas.

Agradeço aos meus orientadores, Luiz Satoru Ochi e José André de Moura Brito. Pela paciência, acessibilidade e por todo conhecimento transmitido.

Agradeço a meus pais, Michel Beltrão e Clara Aparecida, pelo amor, educação e apoio que sempre me deram. Agradeço a todos os meus familiares, principalmente a meus irmãos Ulisses e Lucas.

Agradeço aos Professores do IC/UFF pela dedicação no ensino.

Agradeço àqueles que participaram da minha jornada.

"Quanto melhor é adquirir a sabedoria do que o ouro! E mais excelente, adquirir a prudência do que a prata!"

(Provérbios 16:16)

RESUMO

Neste trabalho aborda-se o Problema do Caixeiro Viajante com Seleção de Hotéis (TSPHS), que é uma variante do Problema do Caixeiro Viajante (TSP), proposta recentemente. No TSPHS, o objetivo é encontrar uma rota para um vendedor visitar todos os clientes e retornar ao ponto de partida, considerando uma restrição associada ao tempo máximo de duração de uma jornada de trabalho. Normalmente, não é possível atender a todos os clientes em uma única jornada, assim, o vendedor deve visitar algum hotel para descansar ao final de cada jornada. A rota é composta por um conjunto de viagens ligadas por hotéis e existe um tempo de visita associado a cada cliente. Os clientes, o ponto de partida, os hotéis e o tempo máximo da jornada de trabalho são dados do problema. O objetivo primário do problema é minimizar o número de viagens e o secundário é minimizar o tempo total de deslocamento.

Nesta dissertação, a abordagem do TSPHS é baseada em uma heurística híbrida, que combina características das metaheurísticas BRKGA e GRASP com procedimentos clássicos da literatura do TSP e algumas de suas variantes, como métodos construtivos e procedimentos de busca local. Nos experimentos conduzidos, o algoritmo proposto foi comparado aos melhores algoritmos da literatura e às melhores soluções encontradas para as 131 instâncias de *benchmark* da literatura. Os resultados do experimento demonstram que o algoritmo proposto é capaz de produzir soluções competitivas frente aos melhores algoritmos da literatura. Mais especificamente, quando comparado aos melhores resultados da literatura, apresenta um *gap* médio de 0,16%, produzindo as melhores soluções para 80% das instâncias; além disso, apresenta soluções melhores que as da literatura em 12 das 131 instâncias testadas.

Palavras-chave: Problema do Caixeiro Viajante Com Seleção de Hotéis, Planejamento de Rotas, Logística e Transporte, Metaheurísticas, Programação Inteira

ABSTRACT

The Traveling Salesperson Problem with Hotel Selection (TSPHS) is addressed in this work, which is a variant of the Traveling Salesperson Problem (TSP), recently proposed. In TSPHS the objective is to find a route for a salesperson to visit all customers and return to the starting point considering a restriction associated with the maximum duration of a working day. Normally it is not possible to visit all customers in a single day, so the salesperson must visit a hotel to rest at the end of each working day. A route consists of a series of trips connected by hotels and there is a service time associated with each client. The customers, the starting point, the hotels and the maximum time of the working day are given in the problem. The primary objective of the problem is to minimize the number of trips and the secondary is to minimize the total travel time.

In this dissertation, the TSPHS approach is based on a heuristic algorithm which combines characteristics of the BRKGA and GRASP metaheuristics with classic procedures from the TSP literature and some of its variants such as constructive methods and local search procedures. In the conducted experiments the proposed algorithm is compared to the best algorithms in the literature and the best solutions found for the 131 benchmark instances in the literature. The results of the experiments demonstrate that the proposed algorithm can produce competitive solutions compared to the best algorithms in the literature. When compared to the best results in the literature, it presents an average gap of 0.16% producing the best solutions for 80% of the instances. In addition it presents better solutions than those found in the literature in 12 of the 131 instances tested.

Keywords: Travelling Salesperson Problem with Hotel Selection, Route Planning, Logistics and Transport, Metaheuristics, Integer Programming

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

GA	Genetic Algorithm.
BRKGA	Biased Random-Key Genetic Algorithm.
ILS	Iterated Local Search.
VND	Variable Neighborhood Descent.
GRASP	Greedy Randomized Adaptative Search Procedure.
NNH	Nearest Neighbour Heuristic.
MFH	Multiple Fragment Heuristic.
CIH	Cheapest Insertion Heuristic.
FIH	Farthest Insertion Heuristic.
TSP	Travelling Salesman Problem.
TSPHS	Travelling Salesperson Problem with Hotel Selection.
t_i	Tempo de visita ao *i*-ésimo cliente.
L	Limite de tempo da jornada de trabalho.
c_{ij}	Tempo gasto no deslocamento da localização *i* para *j*.
D	Número total de viagens.

SUMÁRIO

CAPÍTULO 1 – INTRODUÇÃO

A despeito de todo aparato computacional atualmente disponível, traduzido na forma de computadores dotados de processadores cada vez mais rápidos e com grande quantidade de memória RAM; ainda assim, existem diversos problemas na área da Pesquisa Operacional que constituem grandes desafios, no que concerne à resolução exata. Um exemplo disso é o Problema do Caixeiro Viajante (do termo em inglês, *The Travelling Salesman Problem* – TSP) que é um dos problemas de otimização combinatória mais estudados nas últimas décadas, com diversas variantes conhecidas na literatura [2]. No TSP clássico, dada uma lista de cidades (ou clientes) e as distâncias entre cada par de cidades, o objetivo é encontrar uma rota para o vendedor que começa no depósito (cidade inicial), passa por todas as cidades, retorna ao depósito e cuja soma das distâncias seja mínima.

Na resolução do TSP podem ser consideradas restrições associadas ao transporte, como janelas de tempo para atendimento de clientes [50], frota com veículos homogêneos ou heterogêneos [22], serviços de coleta e entrega [48, 49], múltiplos produtos [13], limite de tempo de duração da rota, múltiplos depósitos [14, 40], múltiplos vendedores [5], redução de emissão de dióxido de carbono (*CO_2 reduction*) [20], parada em hotéis para descanso do vendedor ao fim da jornada de trabalho [56], custo de estadia em hotéis, leis trabalhistas, etc.

Considerando estas características, existem diversas variantes do TSP que são estudadas na literatura, que além de suas aplicações práticas, são problemas tão ou mais difíceis de resolver do que o TSP. A função objetivo considerada nestes problemas pode estar associada, por exemplo, à redução do tempo de atendimento de clientes, ao gasto financeiro de uma empresa com transporte, à qualidade de um serviço e à preservação do meio ambiente.

Tais variantes podem levar em consideração o roteamento de carros, caminhões, helicópteros, drones, etc. Sendo assim, no que diz respeito à resolução dessas variantes, diversos pesquisadores têm concentrado os seus esforços no estudo e desenvolvimento de algoritmos que produzam soluções de alta qualidade e demandem baixo tempo computacional.

O Problema do Caixeiro Viajante com Seleção de Hotéis (do termo em inglês, *The Travelling Salesperson Problem with Hotel Selection* - TSPHS), proposto inicialmente por Vansteenwegen et al. [56], corresponde a uma nova variante do TSP, incorporando restrições associadas ao limite de tempo de serviço por jornada de trabalho e ao tempo gasto com o atendimento de clientes. Por ter sido proposto recentemente, o TSPHS ainda não foi tão explorado como outras variantes do TSP.

Nesta dissertação o TSPHS foi resolvido por meio de abordagens diferentes daquelas já utilizadas na literatura.

No TSPHS existem dois termos que possuem significado específico para o problema.

Viagem (do inglês, *Trips*) – sequência ordenada de clientes que começa e termina em hotéis. O tempo gasto em uma viagem não deve exceder o limite predefinido de tempo da jornada de trabalho; caso contrário, a viagem faz parte de uma solução não viável.

Rota (do inglês, *Route*) – corresponde ao conjunto ordenado de todas as viagens que, juntas, contêm a sequência de todos os clientes visitados. A rota começa e termina no hotel inicial.

A solução para o TSPHS é dada por uma rota, na qual o vendedor deve visitar e atender todos os clientes ao longo das viagens. A visita a um hotel, que ocorre no final de cada viagem, representa o fim de uma jornada de trabalho. O hotel visitado pelo vendedor ao final da viagem *d* deve ser o mesmo hotel no qual o vendedor inicia a viagem *d*+1.

A figura 1 traz um exemplo de solução para o TSPHS. Neste caso, o vendedor precisa fazer 4 viagens para visitar todos os 10 clientes e retornar ao hotel inicial. Pode-se perceber que a terceira viagem começa e termina no mesmo hotel. A primeira viagem (cor azul) começa no hotel 0, passa por 2 clientes e termina no hotel 1. A segunda viagem (cor vermelha) começa no hotel 1, passa por 3 clientes e termina no hotel 2. A terceira viagem (cor verde) começa e termina no hotel 2, e passa por 3 clientes. A quarta e última viagem (cor laranja) começa no hotel 2, passa por 2 clientes e termina no hotel 0.

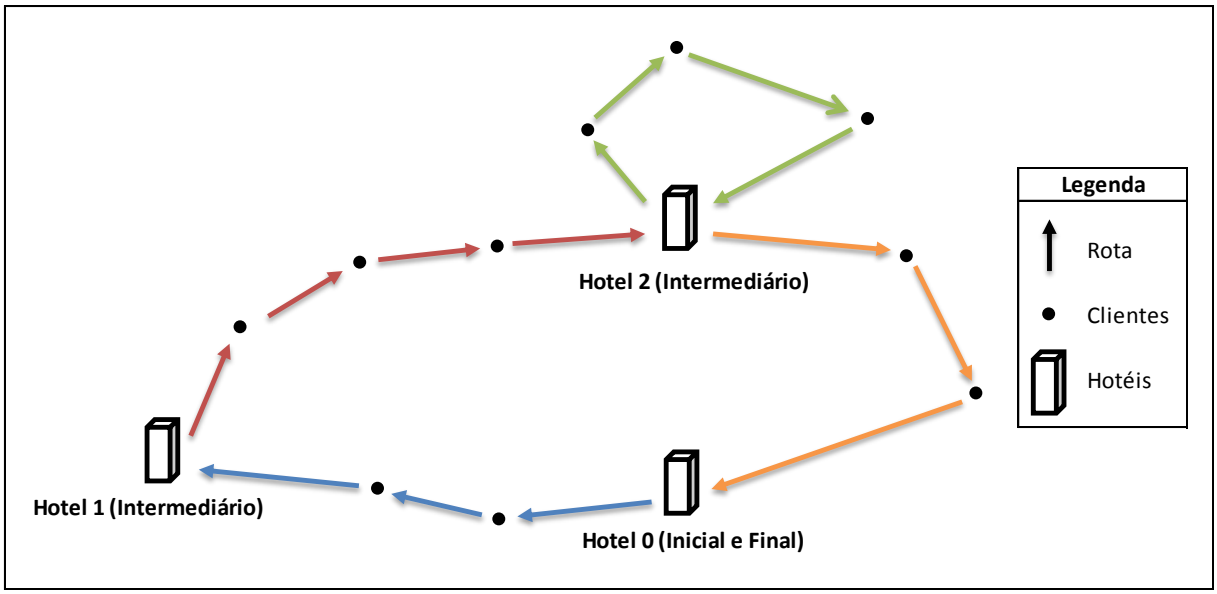

Figura 1: Exemplo de solução para o TSPHS.

No TSPHS o objetivo é minimizar primeiramente o número total de viagens e depois o tempo total de deslocamento do vendedor:

Objetivo Principal – minimizar o número de viagens da rota. O número de viagens corresponde ao número de dias de trabalho de um vendedor, ou seja, representa um custo financeiro e temporal.

Objetivo Secundário – minimizar o tempo total de deslocamento da rota. Este objetivo está indiretamente relacionado ao principal, pois, em geral, uma rota com menor gasto de tempo tende a apresentar um menor número de viagens necessárias.

A escolha para esta ordenação de prioridade dos objetivos é baseada em um problema da vida real. O custo de acrescentar um dia de trabalho para o motorista é maior do que o custo de dirigir alguns quilômetros extras. Diante disso, uma solução com menos viagens é sempre preferível quando comparada com outra solução com mais viagens; ainda que a segunda tenha uma menor distância total.

Deve-se atentar para o fato de que, no TSP clássico, uma solução é composta por uma rota contendo a sequência ordenada de clientes. Deste modo, uma rota do TSP pode ser considerada uma solução para o TSPHS, dada por uma única viagem contendo a visita a todos os clientes, considerando o depósito do TSP como o hotel inicial do TSPHS. No caso do TSPHS, esta solução geralmente é inviável**,** por violar o limite de tempo de serviço da jornada de trabalho. Ainda assim, é possível aproveitar esta sequência de visitas aos clientes. Para tornar uma solução deste tipo viável é necessário dividir esta única "grande viagem" em múltiplas viagens menores, que respeitam o limite de tempo. Esta tarefa pode ser feita particionando a sequência de clientes em múltiplas viagens, por meio da inserção de hotéis em pontos intermediários. Neste sentido, o capítulo 3 será apresentado um procedimento que realiza esta tarefa de forma ótima, mais especificamente, tal procedimento recebe uma sequência ordenada de clientes, e retorna uma rota particionada em viagens que respeitam o limite de tempo de serviço da jornada de trabalho.

Assim como no caso do TSP, o TSPHS também é classificado como um problema NP-difícil [9, 55]. Tal como em outras variantes do TSP, este problema pode ser abordado mediante a aplicação de formulações ou de algoritmos heurísticos. Embora a aplicação de uma formulação garanta a produção de uma solução ótima, sua aplicação está restrita, em geral, a problemas de pequeno porte (da ordem de dezenas de clientes) posto que, para instâncias de porte médio ou grande (da ordem centenas de clientes ou mais), o tempo de execução para a resolução da formulação, mediante aplicação de um algoritmo exato (tipo

Branch and Bound) pode crescer exponencialmente com o tamanho da instância para este tipo de problema.

Já os algoritmos heurísticos produzem soluções de boa qualidade, não necessariamente ótimas, demandando baixo tempo computacional, fato este que motiva abordagens baseadas nesse tipo de algoritmos. Um grande número de algoritmos heurísticos de sucesso para problemas de Otimização Combinatória são baseados em metaheurísticas [56].

Face à complexidade deste problema, propõe-se**,** nesta dissertação, um algoritmo heurístico que foi desenvolvido a partir do estudo das metaheurísticas GRASP e BRKGA e de procedimentos clássicos da literatura, também utilizados para resolver o TSP e algumas de suas variantes. O algoritmo desenvolvido foi aplicado em um conjunto de 131 instâncias da literatura, de diferentes tamanhos (10 a 1002 clientes) e com diferentes quantidades de hotéis (2 a 11 hotéis). Os resultados obtidos por meio dos experimentos foram comparados aos resultados produzidos pelos melhores algoritmos da literatura.

De acordo com os experimentos realizados, o algoritmo proposto foi capaz de produzir resultados semelhantes ou melhores que os da literatura em aproximadamente de 80% dos casos, considerando todas as instâncias. Para as instâncias em que a solução ótima é conhecida, o algoritmo foi capaz de produzir soluções ótimas em 85 de 95 instâncias, ou seja, em quase 90% dos casos. Para as instâncias em que a solução ótima é desconhecida, o algoritmo foi capaz de produzir soluções superiores às melhores soluções conhecidas da literatura em 12 de 36 instâncias.

Na comparação feita entre as soluções produzidas contra os melhores resultados da literatura, verificou-se um *gap* médio inferior a 0,2%, e um *gap* mediano de 0%, demonstrando a alta qualidade das soluções produzidas pelo algoritmo proposto. Estes valores do *gap*, foram calculados a partir do tempo de viagem total da rota, dado pela soma do tempo total de deslocamento da rota com o tempo total gasto no atendimento de clientes.

1.1 OBJETIVOS

Esta dissertação aborda o “Problema do Caixeiro Viajante com Seleção de Hotéis”, que corresponde a uma variante do Problema do Caixeiro Viajante. Os objetivos deste trabalho são descritos a seguir.

Desenvolver um algoritmo heurístico que seja capaz de produzir soluções de alta qualidade em relação aos melhores algoritmos da literatura, demandando tempo de processamento semelhantes aos da literatura.

Realizar experimentos com a abordagem proposta a fim de avaliar sua eficácia e eficiência comparativamente aos resultados da literatura.

Analisar os resultados obtidos nos experimentos realizados a fim de produzir conhecimento útil para o tema abordado.

1.2 ESTRUTURA DA DISSERTAÇÃO

O capítulo 2 apresenta o Problema do Caixeiro Viajante com Seleção de Hotéis, suas principais características, um exemplo de resolução de uma instância, além das principais formulações matemáticas encontradas na literatura. No capítulo 3 é feita uma revisão da literatura sobre o tema abordado, diferenciando o TSPHS de outras variantes do TSP, citando os principais trabalhos, além de comentar as estratégias utilizadas nestes trabalhos para abordagem do problema. No capítulo 4 é feita uma descrição das metaheurísticas nas quais o algoritmo proposto foi inspirado, além do algoritmo, seus procedimentos construtivos, operadores genéticos e procedimentos de busca local. No capítulo 5 estão os resultados dos experimentos computacionais, comparando o algoritmo proposto com outros algoritmos e os melhores resultados encontrados na literatura. Finalmente, no capítulo 6 estão apresentadas as conclusões, possíveis trabalhos futuros e comentários finais.

CAPÍTULO 2 – O PROBLEMA DO CAIXEIRO VIAJANTE COM SELEÇÃO DE HOTÉIS

O presente capítulo traz uma descrição do TSPHS, um exemplo que ilustra a sua resolução e a apresentação das formulações matemáticas encontradas na literatura para a resolução desse problema.

2.1 DEFINIÇÃO DO PROBLEMA

Seja $H = \{0, 1, 2, ..., s\}$ um conjunto de $s+1$ localizações de hotéis e $C = \{s+1, ..., s+n\}$ um conjunto de n localizações de clientes. Cada cliente está associado a um tempo gasto com atendimento ou visita t_i, já os hotéis não apresentam tempo de atendimento ou visita.

O TSPHS pode ser modelado [36] através de um grafo completo $G = (V, A)$, sendo $V = H \cup C$ o conjunto de vértices e $A = \{(i,j) \mid i,j \in V, i \neq j\}$ o conjunto de arestas. As arestas do conjunto A representam os diversos caminhos pelos quais o vendedor pode se locomover entre as localizações (clientes ou hotéis). A cada aresta (i, j) está associado a um peso c_{ij}, conhecido no problema, que corresponde ao tempo necessário de deslocamento da localização i para j (cada localização pode ser um cliente ou um hotel). Sendo assim, o tempo gasto com o deslocamento e atendimento do cliente j, vindo do cliente i, é definido por $c_{ij}+t_j$.

A solução para este problema é dada por uma rota que atenda todos os clientes, sendo tal rota composta por um conjunto de viagens conectadas por hotéis intermediários. Uma viagem d corresponde a um caminho no grafo G, que passa por clientes (localizações de C), começa e termina em hotéis intermediários (localizações de H); sendo que uma viagem d pode começar e terminar no mesmo hotel ou em hotéis diferentes. A viagem $d+1$ começa no hotel que a viagem d terminou. A rota começa e termina no ponto de partida (dado no problema), e este local também pode ser utilizado como hotel intermediário.

Cada viagem contém um subconjunto de clientes, e todo cliente deve ser atendido em exatamente uma das viagens. O tempo limite de uma viagem é dado no problema por TL, intuitivamente interpretado como um dia de trabalho, de modo que, ao final de cada dia em sua jornada de trabalho, o vendedor visite um hotel para descansar. O tempo gasto em uma viagem é igual ao somatório do tempo gasto no deslocamento do vendedor entre clientes, chegada e saída dos hotéis e atendimento dos clientes. Se o tempo de qualquer viagem exceder TL a solução não é viável.

Neste problema, o objetivo principal é minimizar o número de viagens em uma rota (número necessário de dias de trabalho) e o objetivo secundário é minimizar o tempo total de deslocamento da rota.

A figura 2 traz um exemplo do problema, com seis hotéis (s = 5) e dezessete clientes (n = 17). Cada cliente é representado por um ponto (conjunto C). Cada hotel é representado por um quadrado (conjunto H). A rota do vendedor é dada pelo segmento de reta.

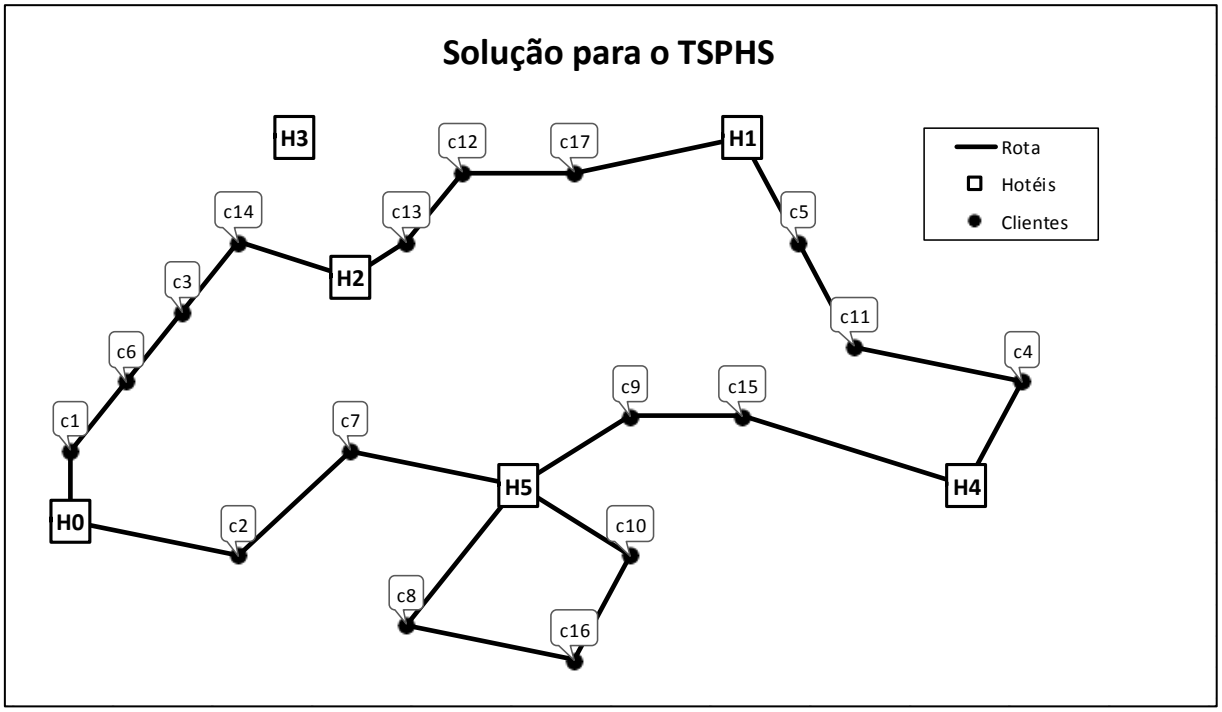

Figura 2: Solução para o TSPHS.

A figura 3 ilustra a mesma solução, desta vez por meio de um vetor contendo a ordem de visita a clientes e hotéis. Pode-se perceber que o hotel H_3 não é visitado nenhuma vez, enquanto o hotel H_5 é visitado duas vezes. Conforme já comentado, a rota começa e termina no ponto de partida H_0.

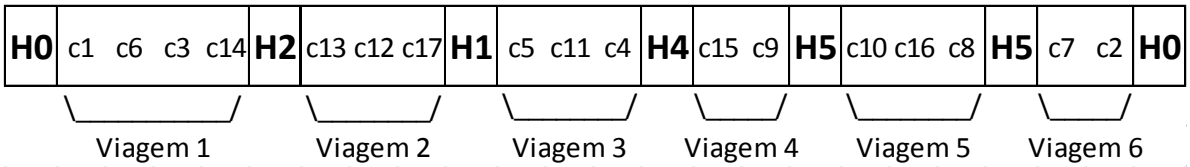

Figura 3: Solução para o TSPHS. Ordem de visita de clientes e hotéis.

A solução apresentada é composta de 6 viagens: V_1, V_2, V_3, V_4, V_5, V_6. Sendo $V_1 = \{H_0, c_1, c_6, c_3, c_{14}, H_2\}$, $V_2 = \{H_2, c_{13}, c_{12}, c_{17}, H_1\}$, $V_3 = \{H_1, c_5, c_{11}, c_4, H_4\}$, $V_4 = \{H_4, c_{15}, c_9, H_5\}$, $V_5 = \{H_5, c_{10}, c_{16}, c_8, H_5\}$, $V_6 = \{H_5, c_7, c_2, H_0\}$.

Para que a solução seja considerada viável é necessário que nenhuma das viagens viole o limite tempo. Ou seja, neste caso, isso significa que todas as restrições a seguir devem ser respeitadas.

$$c_{H0c1} + t_{c1} + c_{c1c6} + t_{c6} + c_{c6c3} + t_{c3} + c_{c3c14} + t_{c14} + c_{c14H2} \leq TL \quad (2.1)$$

$$c_{H2c13} + t_{c13} + c_{c13c12} + t_{c12} + c_{c12c17} + t_{c17} + c_{c17H1} \leq TL \quad (2.2)$$

$$c_{H1c5} + t_{c5} + c_{c5c11} + t_{c11} + c_{c11c4} + t_{c4} + c_{c4H4} \leq TL \quad (2.3)$$

$$c_{H4c15} + t_{c15} + c_{c15c9} + t_{c9} + c_{c9H5} \leq TL \quad (2.4)$$

$$c_{H5c10} + t_{c10} + c_{c10c16} + t_{c16} + c_{c16c8} + t_{c8} + c_{c8H5} \leq TL \quad (2.5)$$

$$c_{H5c7} + t_{c7} + c_{c7c2} + t_{c2} + c_{c2H0} \leq TL \quad (2.6)$$

Cada uma destas restrições representa a soma total do tempo gasto na rota com deslocamento entre clientes e hotéis e atendimento a clientes.

O TSP clássico é um caso especial do TSPHS quando $TL = +\infty$, $t_i = 0$ para todo $i \in C$, e $s = 0$. Assim, o TSPHS é pelo menos tão difícil quanto o TSP e, portanto, também é NP-difícil [9, 55].

2.2 FORMULAÇÕES MATEMÁTICAS

Para melhor entendimento do TSPHS e de suas restrições, nesta seção são apresentadas as duas principais formulações matemáticas encontradas na literatura para o TSPHS. Estas formulações foram aplicadas, separadamente, em 52 instâncias consideradas pequenas na literatura, que contém 2 hotéis e de 10 a 40 clientes.

A primeira formulação foi proposta no trabalho que apresenta o TSPHS em Vansteenwegen et al. [56]. Esta formulação foi aplicada às 52 instâncias pequenas e foi capaz de produzir soluções ótimas para 27 destas instâncias, para um tempo máximo de 2 horas.

A segunda formulação foi proposta em Castro et al. [10]. Esta formulação foi aplicada às 52 instâncias pequenas e foi capaz de produzir soluções ótimas para 27 destas instâncias, para um tempo máximo de 6 horas. Sendo que esta formulação foi capaz de produzir todas as soluções ótimas produzidas pela formulação de Vansteenwegen et al. [56]. Assim, a formulação de Castro et al. [10] foi capaz de produzir resultados melhores que os de Vansteenwegen et al. [56] ao custo de um tempo computacional maior.

2.2.1 FORMULAÇÃO DE VANSTEENWEGEN

Conforme já comentado, o problema possui dois objetivos, sendo o primeiro deles minimizar o número de viagens e o segundo minimizar o tempo total de deslocamento da rota.

Esta formulação leva, em consideração, o segundo objetivo, fixando o número de viagens previamente. Ou seja, foi desenvolvida uma formulação para o TSPHS em que um dos parâmetros de entrada é o número de viagens *D*. Como o valor de *D* para a solução ótima não é conhecido, a formulação precisa ser executada iterativamente até que a solução ótima seja produzida. Na primeira iteração, *D* recebe um valor pequeno (explicado posteriormente), que aumenta em uma unidade a cada iteração, até que alguma solução viável seja produzida. Como *D* recebe valores cada vez maiores, a solução ótima possui um número de viagens igual a *D* para a primeira iteração na qual uma solução viável é produzida.

Assim sendo, a solução ótima para o TSPHS pode ser obtida resolvendo a formulação com o número *D* de viagens fixado previamente, para valores cada vez maiores de *D* até que uma solução viável seja encontrada. Esta estratégia garante a produção da solução ótima em um número finito de iterações. Um valor inicial para *D* pode ser determinado como um número mínimo de viagens (*lower bound*) necessário para realização de todas as visitas, sem levar em consideração o tempo de deslocamento entre clientes e hotéis.

A fórmula para o cálculo do valor inicial para *D* é dada a seguir:

$$D = \sum_{i=s+1}^{s+n} t_i / TL \tag{2.7}$$

O parâmetro *TL* representa o limite de tempo da jornada de trabalho, enquanto t_i representa o tempo de atendimento do cliente *i*.

Sendo assim, para um número *D* de viagens previamente fixado, os autores desenvolveram a seguinte formulação de Programação Inteira para o TSPHS:

Seja x_{ij}^d uma variável binária que recebe o valor 1 se, na viagem *d*, o cliente (ou hotel) *j* é visitado imediatamente após o cliente (ou hotel) *i*, e 0 caso contrário. Seja u_i uma variável inteira que indica a posição do cliente *i* na viagem na qual ele é visitado.

$$\text{minimizar} \qquad \sum_{d=1}^{D} \sum_{i=0}^{s+n} \sum_{j=0}^{s+n} x_{ij}^d c_{ij} \tag{2.8}$$

$$s.a. \sum_{j=0}^{s+n} x_{0j}^1 = \sum_{i=0}^{s+n} x_{i0}^D = 1 \tag{2.9}$$

$$\sum_{h=0}^{s}\sum_{i=0}^{s+n} x_{ih}^{d} = \sum_{h=0}^{s}\sum_{j=0}^{s+n} x_{hj}^{d} = 1\,, \qquad d = 1, \dots, D \tag{2.10}$$

$$\sum_{i=0}^{s+n} x_{ih}^{d} = \sum_{j=0}^{s+n} x_{hj}^{d+1}\,, \qquad d = 1, \dots, D-1;\ h = 0, \dots, s \tag{2.11}$$

$$\sum_{d=1}^{D}\sum_{k=0}^{s+n} x_{ki}^{d} = 1, \qquad i = s+1, \dots, s+n \tag{2.12}$$

$$\sum_{k=0}^{s+n} x_{ki}^{d} = \sum_{l=0}^{s+n} x_{il}^{d}\,, \qquad d = 1, \dots, D;\ i = s+1, \dots, s+n \tag{2.13}$$

$$\sum_{i=0}^{s+n}\sum_{j=0}^{s+n} (x_{ij}^{d}(c_{ij} + t_j)) \le TL\,, \qquad d = 1, \dots, D \tag{2.14}$$

$$u_i - u_j + 1 \le (n-1)\left(1 - \sum_{d=1}^{D} x_{ij}^{d}\right), \qquad i = s+1, \dots, s+n;\ j = s+1, \dots, s+n \tag{2.15}$$

$$x_{ij}^{d} \in \{0, 1\}, \qquad d = 1, \dots, D;\ i = 0, \dots, s+n;\ j = 0, \dots, s+n \tag{2.16}$$

$$u_i \in \{1, \dots, n\}, \qquad i = s+1, \dots, s+n \tag{2.17}$$

A função objetivo (2.8) minimiza o tempo total de deslocamento da rota. O número de viagens *D* não é uma variável de decisão, mas um parâmetro dado para o problema. A restrição (2.9) garante que a rota inicia e termina no hotel 0 (que funciona como depósito). A restrição (2.10) garante que cada viagem começa e termina em algum dos hotéis (0, ..., *s*). A restrição (2.11) garante que, se uma viagem termina em um dado hotel, então a próxima viagem se inicia no mesmo hotel. A restrição (2.12) garante que cada cliente é visitado uma vez e a restrição (2.13) garante a conectividade dentro de cada viagem. A restrição (2.14) está associada ao limite máximo para o tempo de viagem (duração da jornada de trabalho) de cada viagem e a restrição (2.15) é necessária para evitar o surgimento de subrotas. Esta restrição para eliminação de subrotas é inspirada na formulação que Miller-Tucker-Zemlin [38], propuseram para o TSP. Finalmente, as restrições (2.16) e (2.17) apresentam os possíveis valores que as variáveis x_{ij}^{d} e u_i podem assumir.

2.2.2 FORMULAÇÃO DE CASTRO 2013

No trabalho Castro et al. [10], foram propostas duas importantes modificações para a formulação de Vansteenwegen et al. [56].

A primeira modificação é feita na função objetivo, que passa a incorporar um termo associado ao objetivo principal (minimizar o número de viagens). Mais especificamente, a modificação é feita introduzindo uma variável na função objetivo que é multiplicada por um valor *M* (constante de penalização) suficientemente elevado, de modo que a solução que apresenta menor número de viagens é sempre a que apresenta o menor valor da função objetivo. Assim, a função objetivo desta formulação leva em consideração os dois objetivos do problema; primeiramente minimizar o número de viagens e depois, minimizar a soma total das distâncias da rota.

A segunda modificação diz respeito à restrição que elimina subrotas. Conforme já comentado, a formulação original utiliza a restrição de Miller-Tucker-Zemlin para eliminação de subrotas, que é substituída pela restrição de Dantzig-Fulkerson-Johnson para eliminação de subrotas. Segundo Castro et al. [10], com a inclusão desta nova restrição é possível encontrar, a partir da utilização de algum *solver* de otimização, a solução ótima em menos tempo computacional.

Seja x_{ij}^{d} uma variável binária que recebe o valor 1 se, na viagem *d*, o cliente (ou hotel) *j* é visitado imediatamente após o cliente (ou hotel) *i*, e 0 caso contrário.

Seja y^{d} uma variável binária que recebe valor 1, se na viagem *d*, pelo menos um cliente ou hotel é visitado, e 0 caso contrário. Esta variável representa o número de viagens necessárias para completar a rota e está associada ao objetivo principal. *D* equivale ao número máximo de viagens da solução.

$$\text{minimizar} \qquad M\sum_{d=1}^{D} y^{d} + \sum_{d=1}^{D}\sum_{i=0}^{s+n}\sum_{j=0}^{s+n} x_{ij}^{d} c_{ij} \tag{2.18}$$

$$s.a. \sum_{d=1}^{D}\sum_{i=0}^{s+n} x_{ij}^{d} = 1, \qquad j = s+1, \dots, s+n \tag{2.19}$$

$$\sum_{i=0}^{s+n} x_{ij}^{d} = \sum_{i=0}^{s+n} x_{ji}^{d}, \;\; j = s+1, \dots, s+n; \; d = 1, \dots, D \tag{2.20}$$

$$\sum_{h=0}^{s}\sum_{j=0; j\neq h}^{s+n} x_{hj}^{d} = y^{d}, \qquad d = 1, \dots, D \tag{2.21}$$

$$\sum_{h=0}^{s}\sum_{i=0;i\neq h}^{s+n} x_{ih}^{d} = y^{d}, \qquad d = 1,\dots,D \tag{2.22}$$

$$\sum_{i=0}^{s+n}\sum_{j=0}^{s+n}(c_{ij}+t_j)x_{ij}^{d} \leq TL\,, \qquad d = 1,\dots,D \tag{2.23}$$

$$\sum_{j=1}^{s+n} x_{0j}^{1} = 1 \tag{2.24}$$

$$\sum_{i=1}^{s+n} x_{i0}^{d} \geq y^{d} - y^{d+1}, \qquad d = 1,\dots,D-1 \tag{2.25}$$

$$\sum_{i=0}^{s+n} x_{ih}^{d} + y^{d} \geq \sum_{i=0}^{s+n} x_{hi}^{d+1} + y^{d+1}\,, \qquad h = 0,\dots,s;\ d = 1,\dots,D-1 \tag{2.26}$$

$$\sum_{i=0}^{s+n} x_{ih}^{d} - \sum_{i=0}^{s+n} x_{hi}^{d+1} \leq 1 - y^{d+1}, \qquad h = 0,\dots,s;\ d = 1,\dots,D-1 \tag{2.27}$$

$$x_{ij}^{d} \leq y^{d}, \qquad i = 0,\dots,s+n;\ j = 0,\dots,s+n;\ d = 1,\dots,D \tag{2.28}$$

$$y^{d} \geq y^{d+1}, \qquad d = 1,\dots,D-1 \tag{2.29}$$

$$\sum_{i\in K}\sum_{j\in K\setminus\{i\}} x_{ij}^{d} \leq |K| - 1, \qquad K \subset C, \qquad 2 \leq |K| \leq |C| - 1, \qquad d = 1,\dots,D \tag{2.30}$$

$$x_{ij}^{d} \in \{0,1\}, \qquad i = 0,\dots,s+n;\ j = 0,\dots,s+n;\ d = 1,\dots,D \tag{2.31}$$

$$y^{d} \in \{0,1\}, \qquad d = 1,\dots,D \tag{2.32}$$

A função objetivo (2.18) minimiza o custo total (primeiro o número de viagens; segundo o tempo total de viagem). A restrição (2.19) garante que cada cliente é visitado exatamente uma vez. A restrição (2.20) garante a conectividade de cada viagem. As restrições (2.21) e (2.22) garantem que cada viagem começa e termina em algum hotel. A restrição (2.23) está associada ao limite de tempo máximo da jornada de trabalho (*upper bound*). As restrições (2.24) e (2.25) garantem que a rota começa e termina no hotel 0. As restrições (2.26) e (2.27) garantem que, se uma viagem termina em um dado hotel, a próxima viagem se inicia neste mesmo hotel. A restrição (2.28) identifica quando uma viagem está sendo usada, se existe pelo menos um cliente ou hotel sendo visitado, enquanto a restrição (2.29) garante que as viagens ocorrem em dias consecutivos, começando no dia 1. A restrição (2.30), que envolve subconjuntos *K* de conjuntos de clientes de *C*, é a restrição clássica para eliminação de subrotas aplicada a cada viagem. Finalmente, temos as restrições (2.31) e (2.32) binárias para as variáveis x_{ij}^{d} e y^{d}.

CAPÍTULO 3 – REVISÃO DA LITERATURA

Por lidar com planejamento e construção de múltiplas rotas (viagens) e com múltiplos pontos de parada (hotéis), o TSPHS possui características que o tornam similar a diversas variantes do TSP como, por exemplo, o Problema do Caixeiro Viajante Múltiplo (do termo em inglês, *The Multiple Traveling Salesman Problem* – mTSP) [5], o Problema de Roteamento de Veículos (do termo em inglês, *Vehicle Routing Problem* – VRP) [50], Problema de Roteamento de Veículos com Múltiplos Depósitos (do termo em inglês, *Multiple Depot Vehicle Routing Problem* – MDVRP) [14] e os Problemas de Localização-Roteamento (do termo em inglês, *Location-Routing Problems* – LRP) [40].

A seguir, é feita uma breve descrição de cada um dos problemas citados, com o objetivo de identificar suas semelhanças e diferenças em relação ao TSPHS.

Problema do Caixeiro Viajante Múltiplo - Uma generalização do TSP, com *m* vendedores. A solução é dada por um conjunto de *m* rotas que**,** juntas**,** visitam todos os clientes e iniciam e terminam no depósito. O objetivo é minimizar a soma das distâncias percorrida por todos os vendedores.

Problema de Roteamento de Veículos – A solução para este problema é dada por um conjunto de rotas que, juntas, atendem todos os clientes. No entanto, existe uma restrição que impede que um único veículo seja capaz de visitar e atender todos os clientes. A restrição pode ser um limite para a distância que um veículo pode percorrer em uma rota, ou um limite quanto à quantidade de demanda de clientes a ser atendida pelo veículo (capacidade do veículo). O objetivo é usualmente minimizar o custo total das rotas.

Problema de Roteamento de Veículos com Múltiplos Depósitos – similar ao VRP, considerando uma pequena modificação: existem diversos depósitos (pontos de partida); cada um com sua própria frota de veículos.

Problemas de Localização-Roteamento – Diferente do TSP, o LRP não é um problema único e bem definido; antes, ele pode ser pensado como um conjunto de problemas baseados na localização de instalações e planejamento de rotas. Em uma versão simples do LRP, um conjunto de possíveis localizações para instalações é dado, juntamente com um conjunto de clientes que serão visitados. Neste problema é preciso, primeiramente, determinar quais localizações serão usadas como depósitos. Depois disso, os depósitos podem ser utilizados como pontos de partida e chegada para definir as rotas para atender os clientes. O objetivo é minimizar o custo total que

é dado pelo custo das instalações usadas (depósitos utilizados para resolução do problema) somado ao custo das rotas. Uma vez que as instalações tenham sido definidas, o problema se torna um MDVRP. Dito isto, as aplicações do LRP tratam da escolha de localizações em conjunto com o planejamento de rotas, sendo resolvidas por meio da resolução conjunta de dois problemas diferentes.

De igual modo, o TSPHS trata da construção de viagens em conjunto com a escolha de quais hotéis serão visitados; no entanto, no TSPHS há apenas um veículo (vendedor), implicando que os depósitos (hotéis) devem estar conectados por rotas parciais (viagens) limitadas pelo tempo máximo da jornada de trabalho.

O TSPHS foi proposto, inicialmente, por Vansteenwegen et al. [56], sob a motivação de que um vendedor pode não ser capaz de realizar todas as visitas e atendimentos em uma única jornada de trabalho. Após apresentar a definição do problema em questão**,** os autores fizeram uma revisão da literatura, apresentando as diferenças entre o TSPHS e outras variantes do TSP, pois, como já comentado, diversas variantes do TSP possuem características em comum com o TSPHS. Em seguida, um modelo matemático é apresentado assim como diversas instâncias para o TSPHS, adaptadas de outros problemas da literatura, sendo tais instâncias divididas em quatro conjuntos, conforme o modo como foram construídas.

Para resolver o problema os autores propuseram duas heurísticas, denominadas I1LS e I2LS, que fazem uso de procedimentos construtivos diferentes com a mesma busca local, envolvendo clientes e hotéis, para aprimorar as soluções produzidas inicialmente. Ao final, procura-se reduzir o número de viagens, removendo um hotel da rota e mudando a visita a certos clientes para ocorrer em outras viagens. Se a redução é possível, são aplicados novamente os procedimentos de busca local à solução obtida, até que a redução do número de viagens não seja mais possível.

Para avaliar a eficácia das heurísticas propostas, os autores compararam os resultados das duas heurísticas com as soluções produzidas pela formulação matemática, apresentada na subseção 2.2.1, mais especificamente, o *solver* CPLEX (versão 10.0), que foi utilizada para resolver as instâncias do primeiro conjunto de instâncias (que contém entre 48 e 288 clientes e 5 hotéis), considerando um tempo limite de 6 horas. A ideia foi comparar as soluções produzidas pelas heurísticas com as soluções ótimas. Ocorre que o *solver* foi incapaz de produzir a solução ótima para qualquer destas instâncias, considerando o tempo de 6 horas.

Sendo assim, foi criado um segundo conjunto composto por 52 instâncias, com até 40 clientes e apenas 2 hotéis. A quantidade pequena de clientes possibilitou a produção de

soluções ótimas pela formulação matemática. Para este segundo conjunto, considerando um tempo máximo de processamento para o *solver* de 2 horas, a formulação produziu o ótimo em mais de 50% das instâncias.

Os experimentos realizados no primeiro conjunto demonstraram que o I2LS produz soluções melhores que o I1LS; deste modo, apenas o I2LS foi utilizado nos experimentos com os demais conjuntos. O I2LS foi capaz de produzir apenas 11 soluções ótimas para o conjunto 2. O terceiro conjunto contém instâncias maiores (51 a 1002 clientes e 3, 5 e 10 hotéis) e que foram construídas de modo que a solução ótima para cada instância fosse conhecida. A média dos valores dos *gaps* das soluções produzidas pelo I2LS em relação à solução ótima para o conjunto 3 foi de 12,4%. O quarto e último conjunto é composto de instâncias com ótimos desconhecidos; neste caso, as soluções produzidas pelo I2LS são propostas como *benchmark* para estudos futuros e não são comparadas a nenhuma outra solução.

Em 2012, Castro et al. [11] apresentam um algoritmo simples que combina as metaheurísticas GRASP (*Greedy Randomized Adaptive Search Procedure*) [18] e VND (*Variable Neighborhood Descent*) [28, 29]. O GRASP é utilizado para construir uma rota do TSP e então ela é dividida em viagens viáveis para formar uma solução do TSPHS. Por fim, esta solução é aprimorada pelo VND. Os experimentos conduzidos mostram que o algoritmo proposto, denominado VND, gera soluções melhores que o I2LS.

Em 2013, Castro et al. [10] apresentaram um Algoritmo Memético (*Memetic Algorithm* - MA), denominado MA+TS, que combina operadores clássicos de um algoritmo genético [30] com procedimentos de Busca Tabu [24, 25]. Também foram propostas modificações para a formulação de Programação Inteira de Vansteenwegen et al. [56], mais especificamente, a nova formulação matemática está descrita na subseção 2.2.2. O *solver* utilizado (Gurobi 4.6) foi capaz de produzir, com tempo máximo de processamento de 6 horas, soluções ótimas em 47 das 52 instâncias do segundo conjunto. Assim como na formulação, o algoritmo MA+TS foi capaz de produzir as soluções ótimas para todas as 47 instâncias, demandando um tempo de processamento significativamente menor (em média menos de 0,5 segundos e menos de 3 segundos no pior caso). Os experimentos realizados mostraram que o algoritmo é capaz de encontrar soluções de alta qualidade, significativamente melhores que o I2LS e o VND para todos os conjuntos de instâncias.

Em 2014, Sousa & Gonçalves [52] apresentaram dois novos Algoritmos Meméticos sendo cada um deles utilizando um tipo diferente de busca local para aprimorar as soluções do TSHPS. O primeiro, denominado MA_BT, é combinado a uma Busca Tabu semelhante ao que foi proposto em Castro et al. [10]. O segundo algoritmo, denominado AM_RVND, é

combinado ao *Random Variable Neighborhood Descent* (RVND). O RVND é baseado na metaheurística *Variable Neighborhood Search* (VNS) [27], com as estruturas de vizinhança sendo aplicadas de modo aleatório. Os resultados obtidos foram comparados aos do MA+TS de Castro et al. [10].

Em 2015 Castro et al. [9] propuseram um novo algoritmo, capaz de produzir soluções de alta qualidade em pouco tempo de processamento, denominado P-LS. Tal algoritmo constrói uma solução inicial para o problema por meio da aplicação da heurística de Lin-Kernighan [34] tal heurística gera uma rota que passa por todos os clientes e começa e termina no hotel inicial e depois, divide a rota (inserindo os hotéis) em viagens viáveis, aplicando um procedimento inspirado em Prins [43]. A solução é então aprimorada por meio da aplicação de um procedimento baseado no VND. Além disso, também são utilizados dois operadores de perturbação; o primeiro operador de perturbação realoca de modo aleatório um percentual dos clientes (este percentual é dado por um parâmetro θ_1), ou seja, trabalha com a alteração da ordem de visita dos clientes na solução, e o segundo operador de perturbação se baseia em trocar alguns dos hotéis intermediários (a quantidade de hotéis afetados é equivalente a um percentual dado por um parâmetro θ_2). Esta abordagem é competitiva em termos de qualidade de solução, quando comparada aos resultados anteriores da literatura, enquanto apresenta tempos de processamento significativamente menores que os demais algoritmos da literatura, em geral, 10 a 100 vezes menores do que o algoritmo MA+TS, por exemplo. Os experimentos conduzidos mostraram que o tempo de processamento do algoritmo foi inferior a 1 segundo para todas as instâncias com menos de 240 clientes, com tempo máximo de 34,5 segundos para uma instância contendo 1002 clientes. Os autores também desenvolveram uma nova formulação matemática, que difere daquela apresentada em Castro et al. [10], pois esta é baseada em viagens no lugar de arestas (do conjunto *A*).

Em 2015 Sousa et al. [53] propuseram um algoritmo denominado *Client Perturbation Variable Neigborhood Search* (CP-VNS), que foi inspirado na metaheurística VNS. Este algoritmo faz uso da heurística de Lin-Kernighan combinada com o procedimento de divisão utilizado no algoritmo P-LS para geração da solução inicial, combinado com a metaheurística VNS para refinamento da solução. A perturbação utilizada na solução é semelhante ao primeiro operador de perturbação de Castro et al. [9]. Este operador permite realocar um percentual dos clientes de modo aleatório, ou seja, muda a ordem de visita dos clientes na solução. De acordo com os experimentos conduzidos, o CP-VNS produziu soluções competitivas frente às da literatura consumindo pouco tempo de processamento.

Ainda em 2015, Sousa et al. [51] apresentou um estudo acerca de diversas abordagens para o TSPHS envolvendo três novas heurísticas, sejam elas: AMBT, AMRVND e IGVND. Além disso, também propôs uma alteração para o modelo de Programação Inteira de Castro et al. [10]. Os experimentos realizados apresentaram resultados competitivos em relação à literatura, em particular, em algumas instâncias, foram produzidas soluções melhores que as conhecidas da literatura.

Em 2016, Radmanesh et al. [45] fazem um estudo sobre o TSPHS, comparando os resultados de um método exato do *solver* Gurobi 4.5 com o método exato proposto, do *framework* MILP-Tropical. Ambos os algoritmos exatos foram aplicados em 16 instâncias e produziram as soluções ótimas para todas elas. Não foram feitas comparações de resultados com trabalhos prévios, apenas comparação entre um método exato e a abordagem proposta por meio do framework Tropical. Os resultados produzidos não podem ser comparados aos da literatura, pois foram feitas modificações nas instâncias utilizadas.

Lu et al. [36] propuseram um algoritmo híbrido, denominado HDM, combinando Programação Dinâmica e busca local (para soluções viáveis e não viáveis) a um Algoritmo Memético que possui três tipos de cruzamento. A escolha de qual cruzamento será utilizado, a cada iteração, obedece a um mecanismo probabilístico adaptativo inspirado em [37], considerando que a probabilidade de selecionar qualquer um dos cruzamentos é proporcional ao número de soluções de alta qualidade produzidas com aquele cruzamento. Ou seja, o cruzamento que produz as melhores soluções tende a ser o mais utilizado.

O algoritmo foi executado uma única vez para as 131 instâncias de *benchmark* da literatura, demandando um tempo de processamento semelhante ao Algoritmo Memético de Castro et al. [10]. Os resultados encontrados são de alta qualidade e, na maioria dos casos, o HDM é capaz de produzir soluções iguais ou melhores àquelas encontradas na literatura. Até o momento este algoritmo apresenta o maior percentual de soluções ótimas (ou melhor solução encontrada quando a ótima é desconhecida), sendo selecionado para comparação neste trabalho.

Sousa et al. [54] apresentaram um algoritmo, denominado EA-ILS, baseado na metaheurística *Iterated Local Search* (ILS) [35]. A perturbação aplicada à solução se baseia na alteração da ordem de visitação de alguns clientes, com o diferencial de que o percentual escolhido para perturbação é adaptativo - inversamente proporcional ao tamanho da instância (número de clientes), segundo a função *Nonlinear Symmetrical Sigmoidal Function* (4PL), descrita em MyCurveFit (http://mycurvefit.com/). O algoritmo foi executado 30 vezes para as 131 instâncias de *benchmark* da literatura, sendo escolhida a melhor solução após essas

iterações. O EA-ILS foi capaz de produzir soluções de alta qualidade e, em alguns casos, melhorar a qualidade das melhores soluções de algumas instâncias da literatura. O tempo de processamento apresentado foi significativamente menor que o de Castro et al. [10] e o *gap* médio das soluções em relação às melhores da literatura foi um pouco pior.

O TSPHS também possui variantes já abordadas na literatura, como o Problema do Caixeiro Viajante Múltiplo com Seleção de Hotéis (do termo em inglês, *The Multiple Travelling Salesperson Problem with Hotel Selection* – mTSPHS) [12] e o Problema do Caixeiro Viajante com Múltiplas Janelas de Tempo e Seleção de Hotéis (do termo em inglês, *The Travelling Salesman Problem with Multiple Time Windows and Hotel Selection* – TSP-MTWHS) [3].

Finalmente, uma revisão sistemática da literatura do TSPHS, recentemente publicada, pode ser encontrada em de Sousa et al. [17].

CAPÍTULO 4 – ALGORITMO PROPOSTO

O presente capítulo traz a descrição de um algoritmo heurístico híbrido proposto neste trabalho para resolver o TSPHS. Tal algoritmo combina procedimentos conhecidos da literatura aplicados à resolução do TSP e VRP como, por exemplo, métodos construtivos e de busca local associados ao TSP clássico, além de um algoritmo para inserção de hotéis, baseado em um procedimento que é utilizado em variantes do VRP para partição de rotas. Além disso, o algoritmo utiliza conceitos das metaheurísticas Algoritmo Genético de Chaves Aleatórias Viciadas (do termo em inglês, *Biased Random-Key Genetic Algorithm* - BRKGA) [26] e GRASP (*Greedy Randomized Adaptive Search Procedures*) [18].

Conforme já apresentado no capítulo 3, já existem algoritmos na literatura do TSPHS baseados nas metaheurísticas GRASP, ILS, MA, TS, VND e VNS. Até o momento, de acordo com a pesquisa bibliográfica realizada durante o desenvolvimento desta dissertação, não foram encontrados trabalhos na literatura que utilizem o BRKGA para resolver o TSPHS. O algoritmo aqui desenvolvido faz uso dos operadores genéticos do BRKGA. Além disso, existe a motivação de desenvolver um novo algoritmo heurístico para o TSPHS baseado em metaheurísticas que utiliza uma estratégia alternativa àquela usada nos demais algoritmos da literatura. Esta estratégia se baseia, principalmente, na representação das soluções por meio de cromossomos que não contém informações referentes a hotéis. A explicação detalhada assim como as implicações desta estratégia estão descritas na seção 4.3.

Nas subseções 4.1 e 4.2 são apresentadas as descrições das metaheurísticas BRKGA e GRASP respectivamente. A heurística proposta para resolver o TSPHS será apresentada na subseção 4.3.

4.1 METAHEURÍSTICA BRKGA

O BRKGA é uma metaheurística evolutiva, baseada no Algoritmo Genético com representação de chaves aleatórias de Bean [4], que pode ser aplicada a diversos problemas de otimização, representando suas soluções através de um vetor de chaves aleatórias.

Cada chave aleatória é um número real, gerado aleatoriamente, no intervalo contínuo [0,1). Um vetor de chaves aleatórias pode, então, ser transformado em uma solução para o problema de otimização abordado por meio da aplicação de um decodificador, que corresponde a um procedimento definido de forma específica para cada problema. Em linhas gerais, o decodificador recebe um vetor de chaves aleatórias, transforma este vetor em uma

solução para o problema de otimização em questão**,** e calcula o custo desta solução de acordo com uma função objetivo.

O vetor de chaves aleatórias é considerado como um indivíduo (representado por um cromossomo) dentro de uma população de tamanho p e as chaves aleatórias representam suas n características. Assim, a população de p indivíduos e n características é representada por uma matriz de dimensão p x n.

O funcionamento do BRKGA ocorre por meio da aplicação de três operadores genéticos (Seleção, Cruzamento e Mutação) na população, sendo tais operadores genéticos aplicados para gerar as populações das gerações seguintes.

Seleção: os p cromossomos da população (vetores de chaves aleatórias) são particionados em dois conjuntos: Conjunto Elite (de tamanho $p_e < p/2$) que contém os melhores cromossomos segundo o valor da função objetivo e o Conjunto Não Elite (de tamanho $p - p_e$) que contém os demais cromossomos. O Conjunto Elite é copiado, sem alteração, para a população que será considerada na próxima geração. O objetivo deste operador é garantir que as melhores soluções não sejam perdidas.

Mutação: gera uma quantidade p_m de cromossomos mutantes, que são novos vetores de chaves aleatórias. Este operador é importante por adicionar soluções novas à população, geradas de forma aleatória. Ou seja, este operador adiciona variabilidade à população.

Cruzamento: dois cromossomos (considerados pais) da população são selecionados de maneira aleatória e combinados para gerar um novo cromossomo (considerado filho), sendo um cromossomo do Conjunto Elite (cromossomo a) e outro do Conjunto Não Elite (cromossomo b). Os cruzamentos são realizados sucessivamente até que determinado número ($p - p_e - p_m$) de cromossomos filhos sejam gerados. Cada chave aleatória do cromossomo filho tem probabilidade p_c (maior que 0,5) de ser igual à chave aleatória do cromossomo a e probabilidade $1 - \rho_c$ do cromossomo b. Este processo ocorre com reposição, sendo assim, um indivíduo pode ter mais de um filho por geração.

Depois de aplicar os três operadores genéticos à primeira população obtém-se, na segunda geração, uma nova população formada pelo Conjunto Elite da população anterior, cromossomos mutantes e cromossomos filhos. Os três operadores genéticos são aplicados a cada geração, até que uma condição de parada seja satisfeita. A figura a seguir descreve o funcionamento do BRKGA, considerando duas gerações seguidas.

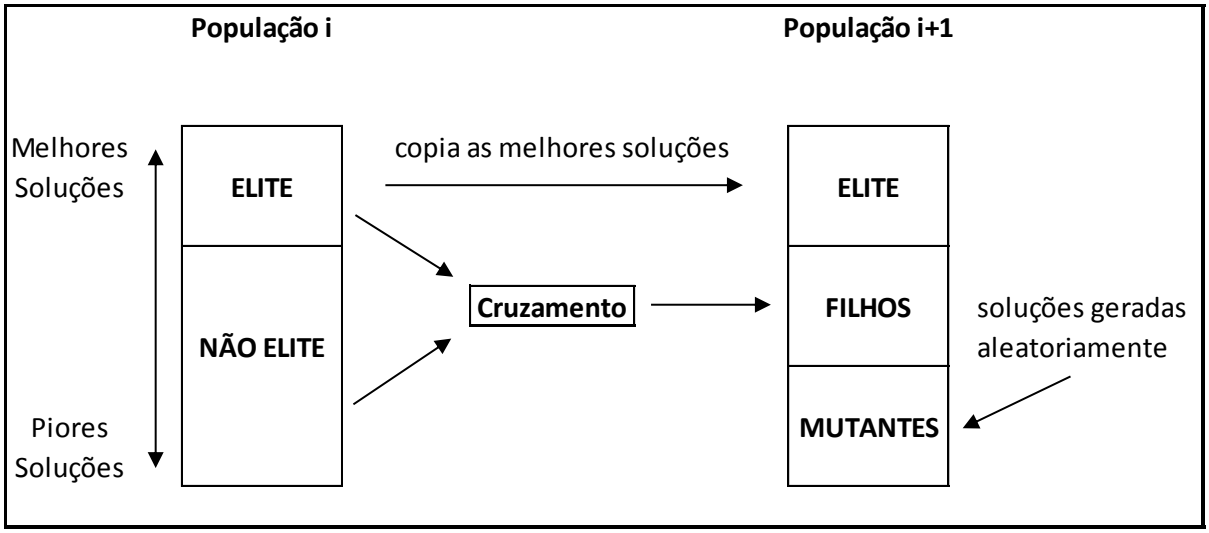

Figura 4. Transição da geração i para geração i+1 em um BRKGA.

4.2 METAHEURÍSTICA GRASP

A metaheurística GRASP (*Greedy Randomized Adaptive Search Procedure*), proposta por Feo & Resende [18], combina procedimentos de construção e busca local. Trata-se de uma estratégia *multi-start* que gera, durante *q* iterações, diversas soluções viáveis e de boa qualidade, tomando-se, dentre estas, a com menor valor de função objetivo (no caso de um problema de minimização).

O GRASP é composto por duas fases. A primeira fase, construtiva, gera uma solução viável inicial e a segunda, busca local, aprimora a solução produzida. A seguir, é apresentado um pseudocódigo do GRASP.

Algoritmo 1: GRASP()

```
1. INÍCIO
2. f* ← +∞;
3.  PARA q←1 ATÉ numero_iteracoes FAÇA
4.   X ←ConstruirSolucao();
5.   X '←BuscaLocal(X);
6.   SE (f(X') < f*) ENTÃO
7.    X * ← X';
8.    f* ← f(X');
9.   FIM-SE
10. FIM-ENQUANTO
11. FIM
12. RETORNA X *;
```

4.2.1 PROCEDIMENTO CONSTRUTIVO

O objetivo nesta fase é construir, iterativamente, uma solução inicial viável, fazendo uso de um componente probabilístico.

Uma possibilidade para esta fase é a utilização de um algoritmo guloso randomizado que, em cada iteração, adicionada um elemento a uma solução *X*, chamada de parcial. Mais especificamente, pensando em um problema de minimização, o elemento a ser adicionado à solução *X* deve ser escolhido de uma Lista de Candidatos (LC) ordenada, de acordo com o incremento no custo de adicioná-lo à solução parcial. Em outras palavras, a LC é composta por todos os elementos que podem ser adicionados à solução parcial naquela iteração, ordenados de acordo com o custo de adicioná-los à solução parcial. Ocorre que, geralmente, é mais vantajoso levar em consideração apenas elementos de custo mais baixo. Deste modo, é possível restringir a LC para conter apenas os melhores elementos de cada iteração.

A Lista Restrita de Candidatos (do termo em inglês, *Restricted Candidate List* - RCL) é composta pelo conjunto dos melhores elementos para aquela iteração, segundo um critério guloso.

A RCL pode ser definida em função dos *k* melhores candidatos, ou seja, *k* elementos que**,** ao serem inseridos à solução parcial**,** produzem os menores acréscimos (problema de minimização).

Outra abordagem é associar o tamanho da RCL ao custo dos candidatos, e seu tamanho varia ao longo das iterações do método construtivo. Seja *g(i)* o custo de adição do elemento *i* à solução em construção. Sejam g_{min} o menor e g_{max} o maior custo de adição, e $\bar{C}$ é o conjunto de todos os elementos fora da solução parcial:

$$g_{min} = \min_{i \in \bar{C}} g(i), \qquad g_{max} = \max_{i \in \bar{C}} g(i) \tag{9}$$

Neste caso, o tamanho da RCL está associado a um parâmetro $\alpha \in [0,1]$. A RCL é composta por todos os candidatos cujo custo *g(i)* é inferior a um valor μ, sendo $\mu = g_{min} + \alpha(g_{max} - g_{min})$. Na medida em que o parâmetro α recebe um valor próximo de 0, o método construtivo se assemelha a um algoritmo guloso (RCL contém apenas 1 candidato). Além disso, valores mais próximos de 1 para o α, fazem o algoritmo gerar soluções aleatórias mais variadas (RCL contém todos os candidatos sempre). A escolha de qual elemento da RCL será adicionado à solução parcial é feita de modo aleatório, com probabilidade de seleção igual para todos os candidatos. Este é o componente aleatório do GRASP, que garante variabilidade quanto às soluções geradas [19]. Após selecionar um

elemento e adicioná-lo à solução parcial, a RCL é atualizada com novos candidatos e assim por diante. Ao fim deste procedimento obtém-se uma nova solução viável.

O componente aleatório permite que repetidas execuções do procedimento construtivo produzam diferentes soluções iniciais diferentes para a fase de busca local. A seguir, é apresentado um pseudocódigo para um procedimento construtivo do GRASP.

Procedimento 1: ConstruirSolucao(*X*)

```
1. INÍCIO
2. X = Ø;
3. construir lista de candidatos: LC();
4. ordenar elementos da lista de candidatos em ordem crescente de custo;
5.  ENQUANTO (solução não completa) FAÇA
6.    RCL ←ConstruirRCL();
7.    v = selecionarElementoAleatoriamente(RCL);
8.    X = X ∪ {v}
9.  FIM-ENQUANTO
10.FIM
9. RETORNA X;
```

Na Literatura também é possível encontrar outros procedimentos construtivos [19], como o *sample greedy* e *random plus greedy*.

4.2.2 PROCEDIMENTO DE BUSCA LOCAL

Conforme já comentado, a solução construída na primeira fase corresponde, geralmente, a um ótimo local de qualidade apenas razoável. Assim sendo, aplica-se a busca local com objetivo de aprimorar as soluções geradas. O procedimento se baseia em tentar transformar a solução atual em uma solução melhor por meio de uma única operação de modificação, denominada movimento. A busca termina quando não é mais possível transformar a solução atual em uma melhor [19].

A vizinhança de uma solução *X* é dada pelo conjunto de soluções *N(X)*. Cada solução $X' \in N(X)$ é resultante da aplicação em *X* de um movimento. Ou seja, *N(X)* é composto por todas as soluções que diferem em 1 único movimento de *X*, denominadas vizinhas de *X*.

Normalmente, duas soluções vizinhas *X'* e *X''* são parecidas, diferindo em poucos elementos. [47]

Uma solução *X**, que corresponde a um ótimo local para uma dada vizinhança *N*, apresenta custo menor (problema de minimização) que todas as outras soluções pertencentes a esta vizinhança, ou seja, $f(X^*) \leq f(X), \forall X \in N(X^*)$. Procedimentos de busca local se baseiam em explorar vizinhanças de uma determinada solução iterativamente, até que não seja mais

possível aprimorar a solução. Deste modo, em geral, quanto maior a vizinhança explorada, maiores são as chances de melhorar a solução.

Um pseudocódigo para uma busca local, de um problema de minimização, é apresentado a seguir:

Procedimento 2: buscaLocal(X)

1. **INÍCIO**
2. flag = 1;
3. **ENQUANTO** (flag == 1) **FAÇA**
4. flag ← 0;
5. encontre $X' \in N(X)$ tal que $f(X') \leq f(X)$;
6. **SE** ($f(X') < f(X)$) **ENTÃO**
7. $X \leftarrow X'$;
8. $f(X) \leftarrow f(X')$;
9. flag = 1;
10. **FIM-SE**
11. **FIM-ENQUANTO**
12. **FIM**
13. **RETORNA** X;

4.3 HEURÍSTICA PROPOSTA: BRKGA-LS

Conforme comentado anteriormente, o algoritmo proposto nesta dissertação combina heurísticas do TSP com um procedimento que particiona de forma ótima a sequência de clientes em viagens viáveis do TSPHS. De acordo com Castro et al. [10], a seleção de hotéis tem grande impacto na qualidade da solução final pois determina a orientação da rota e a forma como as viagens podem ser construídas. Deste modo, a construção de uma solução para o problema passa por dois tipos de decisão, a seleção de hotéis e a ordenação de clientes.

Para diversos algoritmos da literatura (por exemplo MA+TS [10], P-LS [9], HDM [36], EA-ILS [54]) existe a necessidade de tratamento de soluções inviáveis. Estes algoritmos fazem uso de alguma constante de penalização para soluções inviáveis durante a execução da busca local, adicionando um custo a viagens que violam o limite *TL* associado ao tempo máximo da jornada de trabalho. Geralmente, estes algoritmos utilizam o procedimento de partição apenas para inserção de hotéis na solução inicial.

No lugar de trabalhar com uma população de soluções do TSPHS, o BRKGA-LS trabalha com uma população de cromossomos. Neste caso, cada cromossomo contém apenas a sequência de clientes, o que permite a simplificação do problema para tomada de decisão apenas sobre a ordenação dos clientes, sem levar em consideração o limite *TL* ou a seleção dos hotéis. A vantagem de utilizar esta abordagem é que qualquer cromossomo da população sempre está associado a uma solução viável do TSPHS e não há a necessidade de tratamento

de soluções inviáveis. Além disso, o espaço de soluções do problema é reduzido, pois considera-se apenas a sequência de clientes sem levar em consideração a seleção de hotéis. Por fim, também não há necessidade de utilizar estruturas de vizinhança associadas a seleção de hotéis durante a fase de busca local. O procedimento de partição é apresentado a seguir.

Cada indivíduo da população é representado por um cromossomo, que corresponde a um vetor *X*, de tamanho *n*, contendo a sequência ordenada de todos os clientes, onde cada entrada do vetor é um número inteiro que indica o índice de um cliente (podendo ser interpretado como uma solução para o TSP). O cromossomo *X* pode ser mapeado em uma solução viável para o TSPHS, por meio de um procedimento de partição. A ordem de visitação dos clientes pelo vendedor na solução do TSPHS corresponde a mesma ordem de clientes do vetor *X* fornecido na entrada do procedimento. A partição é feita de forma ótima, assim a busca pela solução ótima para o TSPHS se baseia apenas em encontrar a ordenação correta de clientes. O procedimento está descrito em [9] e foi inspirado no trabalho de Prins [43], que usou um procedimento parecido para o VRP, aqui adaptado para o TSPHS. Este é um procedimento determinístico, sendo assim, a qualidade da solução do TSPHS produzida, depende exclusivamente do vetor *X*.

No procedimento de partição, dada uma sequência ordenada de clientes, um grafo auxiliar Q é construído utilizando todos os clientes e hotéis dados no problema. Os arcos deste grafo representam as viagens e os vértices são pontos (hotéis) de chegada e partida de cada viagem. Cada aresta está associada a um peso que corresponde ao custo daquela viagem. O custo de uma viagem é dado pela soma total do tempo de deslocamento do vendedor, mais o tempo de atendimento aos clientes naquela viagem.

A primeira viagem começa no hotel inicial e a última viagem termina no mesmo hotel, após visitar todos os clientes. Com o objetivo de minimizar o número de viagens feitas (arcos percorridos), um valor *W* arbitrariamente grande é adicionado ao peso de cada viagem viável.

Seja *X* o vetor associado à sequência ordenada de *n* clientes, e seja x_i o número do cliente na *i*-ésima posição desta sequência, sendo $i = 0, \dots, n-1$.

O grafo auxiliar construído é denotado por $Q = (V, E, Z)$, sendo *V* o conjunto de $(s+1)(n+1)$ vértices, *E* é o conjunto de arcos e *Z* denota os pesos associados a cada arco de *E*.

O conjunto *V* consiste em dois tipos de vértices, quais sejam: vértices v_i^h com $i < n$ que denotam chegar no hotel *h* antes de visitar o cliente x_i e um vértice v_n^h que denota chegar no hotel *h* depois de visitar todos os clientes.

Um arco $(v_i^p, v_{j+1}^q) \in E$ representa uma viagem que começa no hotel *p*, visita todos os clientes da *i*-ésima até *j*-ésima posição e termina no hotel *q*. O peso $z_{i\ j+1}^{p\ q} \in Z$ associado a qualquer arco (v_i^p, v_{j+1}^q) representa a soma do tempo gasto no deslocamento do vendedor e atendimento de clientes da viagem de v_i^p até v_{j+1}^q. Caso o peso do arco seja superior ao limite *TL* da jornada de trabalho, este arco representa uma viagem que não pode fazer parte de uma solução viável, sendo atribuído a este peso infinito. Seja *X* um vetor, x_h é a *h*-ésima entrada deste vetor e contém o índice do *h*-ésimo cliente visitado na rota. O peso de um arco é dado pela equação a seguir:

$$z_{i\ j+1}^{p\ q} = dist(H_p, x_i) + \sum_{h=i}^{j-1} dist(x_h, x_{h+1}) + dist(x_j, H_q) + \sum_{h=i}^{j} t_h + W \qquad (4.1)$$

O algoritmo de *Dijkstra* [1] é aplicado sobre o grafo auxiliar (presente na figura 6) para encontrar o caminho de menor custo de v_0^0 para v_n^0. Este caminho corresponde ao trajeto que o vendedor faz começando no hotel inicial, visitando todos os clientes e, terminando no hotel inicial. Deste modo, o procedimento retorna o caminho que minimiza o número de viagens e a distância percorrida ao longo deste caminho, sem violar a restrição de tempo das viagens [9]. Esse procedimento foi adotado para mapear um vetor *X* contendo a sequência ordenada de clientes em uma solução viável para o TSPHS.

A seguir é apresentado um exemplo de aplicação deste procedimento em uma instância com 2 hotéis (s = 1) e n = 4 clientes. Neste caso, o cromossomo contendo a sequência de clientes corresponde a $X = \{c_2, c_3, c_1, c_4\}$. O procedimento é aplicado inserindo os hotéis necessários entre clientes para produzir a solução viável para o TSPHS, qual seja: $\{H_0, c_2, c_3, c_1, H_1, c_4, H_0\}$. A solução sempre começa e termina no hotel inicial H_0, sendo que, neste caso, foi necessário inserir apenas um hotel intermediário. Deste modo, a solução produzida é composta por 2 viagens $V_1 = \{H_0, c_2, c_3, c_1, H_1\}$ e $V_2 = \{H_1, c_4, H_0\}$. Segue a figura contendo o exemplo.

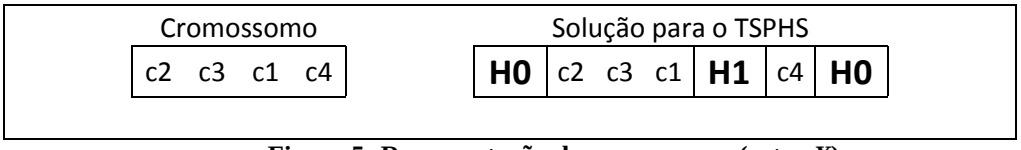

Figura 5: Representação do cromossomo (vetor *X*).

Em outras palavras, a primeira viagem se inicia em H_0, passa pelos clientes da posição 0 até a posição 2 do vetor *X* (que são c_2, c_3, c_1) e finaliza a primeira viagem em H_1. A segunda viagem se inicia em H_1, visita o cliente na posição 3 do vetor *X* (que é c_4) e finaliza a segunda viagem em H_0. No grafo auxiliar da figura 6 cada aresta corresponde a uma viagem: (v_0^0, v_3^1)

representa a primeira viagem e (v_3^1, v_4^0) representa a segunda viagem. Assim, o vértice v_0^0 indica o início da primeira viagem, v_3^1 indica o final da primeira viagem e o início da segunda viagem. O vértice v_4^0 indica o final da segunda viagem. A figura a seguir apresenta o grafo auxiliar no qual o algoritmo de *Dijkstra* é aplicado para encontrar o menor caminho entre v_0^0 e v_4^0.

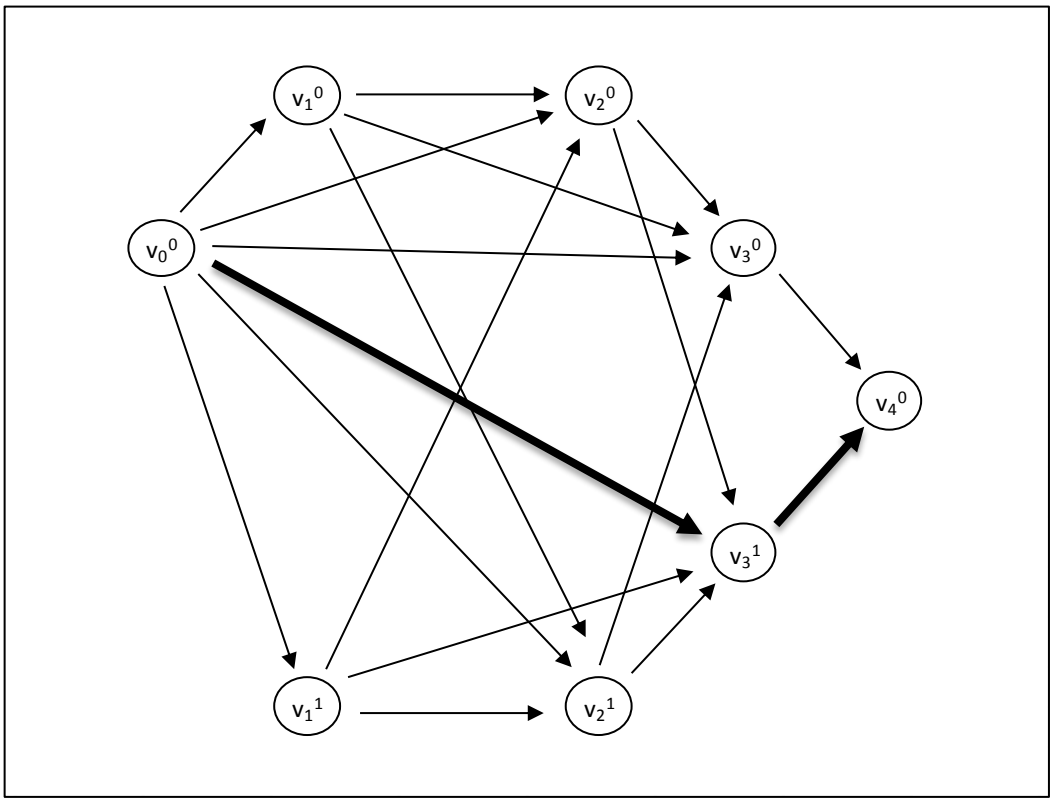

Figura 6: Grafo auxiliar utilizado para produzir uma solução para o TSPHS.

Este procedimento pertence a uma categoria de procedimentos de divisão conhecida como *order-first split-second* segundo a literatura do VRP. Uma pesquisa recente contendo este e outros procedimentos de divisão foi feita em Prins et al. [44].

O algoritmo aqui desenvolvido, denominado BRKGA-LS, recebe, em sua estrutura de entrada, a instância (contendo as localizações dos clientes e hotéis, número *n* de clientes, número *s* de hotéis e tempo limite de viagem *TL*), tamanho *p* da população *P*, tamanho p_e do conjunto elite P_E, tamanho p_m do conjunto de mutantes P_M, probabilidade p_{ls} de aplicação da busca local e o limite *TP* para o tempo de processamento (critério de parada). Existem, ainda, outros dois parâmetros: α e ρ_c. Os valores desses dois parâmetros não são informados na estrutura de entrada do algoritmo, pois eles são calibrados pelo próprio algoritmo ao longo das gerações, com base em um mecanismo probabilístico descrito na seção 5.2.

O quadro a seguir traz o pseudocódigo completo do algoritmo proposto para o TSPHS.

Algoritmo 2: BRKGA-LS(*instância, p, p_e, p_m, p_{ls}, TP*)

1. **INÍCIO**
2. f* ← +∞;
3. Inicialização da matriz de distâncias;
4. Gerar a população inicial *P* com execuções de heurísticas construtivas + busca local;
5. **ENQUANTO** (critério de parada não satisfeito) **FAÇA**
6. Aplique alg. de *Dijkstra* e avalie o custo f(X_i) para cada vetor X_i em *P*;
7. Particione *P* em dois conjuntos: P_E e P_{NE};
8. Inicialize a população da nova geração: $P_{nova} \leftarrow P_E$;
9. Gere conjunto de mutantes P_M, com p_m execuções do GRASP;
10. Adicione P_M à população da próxima geração: $P_{nova} \leftarrow P_{nova} \cup P_M$;
11. rngLS = gerarNumeroAleatorio(1, $p - p_e - p_m$);
12. **PARA** i ← 1 **ATÉ** ($p - p_e - p_m$) **FAÇA**
13. Escolha uma solução X_i aleatoriamente de P_E;
14. Escolha uma solução X_j aleatoriamente de P_{NE};
15. **SE** (i MOD 3 = 0) **ENTÃO** X_{filho} = cruzamento1(X_i, X_j, ρ_c)
16. **SE** (i MOD 3 = 1) **ENTÃO** X_{filho} = cruzamento2(X_i, X_j, ρ_c)
17. **SE** (i MOD 3 = 2) **ENTÃO** X_{filho} = cruzamento3(X_i, X_j, ρ_c)
18. **SE** (i = rngLS) **ENTÃO**
19. Aplique o procedimento de busca local ao cromossomo gerado: VND_4opt(X_{filho})
20. **SENÃO**
21. Aplique o procedimento de busca local ao cromossomo gerado com probabilidade p_{ls}: RVND(X_{filho})
22. **FIM-SE**
23. Adicione o filho X_{filho} à população da próxima geração: $P_{nova} \leftarrow P_{nova} \cup \{X_{filho}\}$;
24. **FIM-PARA**
25. Atualize a população: $P \leftarrow P_{nova}$;
26. Ache a melhor solução X_{nova} em *P*;
27. **SE** (f(X_{nova}) < f*) **ENTÃO**
28. $X^* \leftarrow X_{nova}$;
29. f* ← f(X_{nova});
30. **FIM-SE**
31. **FIM-ENQUANTO**
32. **FIM**
33. **RETORNA** X^*;

Concluindo, um indivíduo da população é sempre representado por um cromossomo contendo apenas a ordem de visita aos clientes, sem possuir quaisquer informações referentes aos hotéis visitados. Ou seja, os cromossomos da população não apresentam informações acerca dos hotéis, o que implica que um cromossomo filho produzido pelo cruzamento não herda características dos cromossomos pais referentes a visita de hotéis; herda, apenas,

características referentes à ordenação de visita aos clientes. Uma consequência disso é que a solução para o TSPHS, associada ao cromossomo filho pode conter visitas a hotéis diferentes dos hotéis visitados nas soluções associadas aos cromossomos pais. Essa característica é possível por causa do procedimento baseado no algoritmo de *Dijkstra*. A figura a seguir ilustra esta questão.

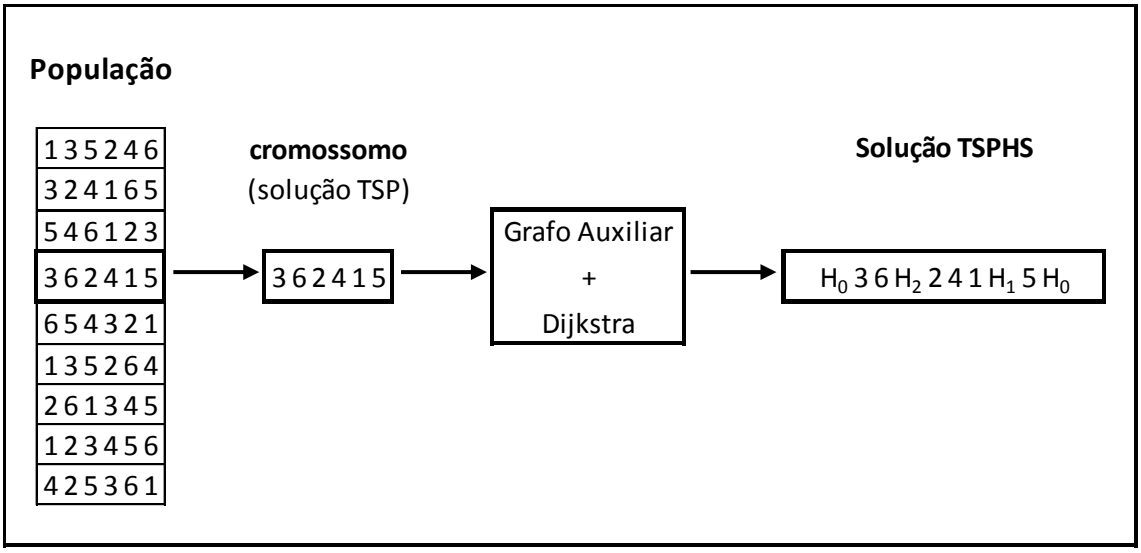

Figura 7: Cromossomo da população sendo mapeado em uma solução para o TSPHS.

4.3.1 POPULAÇÃO INICIAL

Conforme já explicado, uma solução para o TSPHS pode ser obtida a partir de uma sequência ordenada de clientes, por meio da aplicação de um procedimento baseado no algoritmo de *Dijkstra*. Por este motivo, um cromossomo da população pode ser interpretado como uma solução para o TSP. Assim, existe uma correspondência entre uma solução para o TSP e o TSPHS.

Uma solução para o TSPHS pode ser produzida em duas etapas. Primeiro gera-se uma rota por meio de alguma heurística do TSP, composta por uma única viagem, ignorando o limite de tempo *TL* da jornada de trabalho. Depois, a rota é particionada em mais viagens por meio do procedimento baseado no algoritmo de *Dijkstra*. Deve-se levar em consideração que, o hotel inicial (H_0) é considerado como o ponto de partida (depósito) para a rota do TSP.

Sendo assim, todas as heurísticas construtivas aqui utilizadas (linha 4, algoritmo 2) geram soluções para o TSP, que correspondem a cromossomos e serão posteriormente transformadas em soluções para o TSPHS (pelo procedimento de partição) para fins de cálculo do custo segundo a função objetivo. Esta transformação ocorre durante a ordenação e cálculo do custo das soluções associadas aos indivíduos da população (linha 6, Algoritmo 2). Todas as heurísticas construtivas são aplicadas em um grafo G^* contendo a localização dos n

clientes e do hotel inicial (que funciona como o deposito no TSP) do grafo *G*. A população inicial é gerada por execuções de heurísticas construtivas, descritas a seguir:

HC1: Heurística do Vizinho Mais Próximo (*Nearest Neigbourhood Heuristic* - NNH) [7]. Trata-se de uma heurística gulosa que constrói uma solução adicionando elementos iterativamente à solução parcial, sendo cada cliente representado por um vértice do grafo *G**. Inicialmente, a solução parcial é composta de apenas um vértice e, sucessivamente, adiciona-se um vértice (ainda não escolhido) à solução parcial até que ela contenha todos os *n* vértices. O vértice seguinte a ser adicionado é aquele que está mais próximo do último vértice adicionado à solução. Neste caso, o cliente inicial é escolhido aleatoriamente. A figura a seguir apresenta a aplicação da NNH (considerando já adicionados 10, 50 e 100 clientes à solução) em uma instância formada por um conjunto de 100 pontos (que representam 100 clientes).

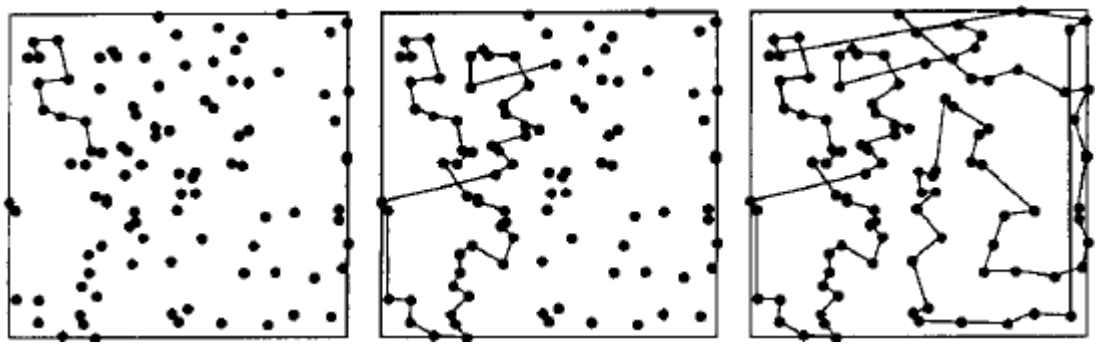

Figura 8. Rota construída pela heurística NNH. Imagem retirada de Bentley [7].

HC2: Heurística do Vizinho Mais Distante. Semelhante a NNH, no entanto, com uma diferença: o elemento escolhido a cada iteração é o cliente não visitado que está mais distante.

A solução é construída adicionando os vértices iterativamente à solução parcial até que ela contenha todos os vértices. Inicialmente, a solução parcial é composta de apenas um vértice, escolhido aleatoriamente. A cada iteração, um vértice (ainda não escolhido) é adicionado à solução parcial, sendo tal vértice aquele que está mais distante do último vértice adicionado.

HC3: Heurística de Múltiplos Fragmentos (*Multiple Fragment Heuristic* - MFH) [7]. Assim como a NNH, a MFH é uma heurística gulosa. Os clientes são representados por vértices de um grafo e as arestas são as ligações entre eles. O peso de cada aresta do grafo é a distância entre os dois clientes ligados por ela. No grafo existe um total de *n* vértices e $n * (n-1)/2$ arestas. No início da execução do MFH as arestas são ordenadas em ordem crescente, de acordo com seus pesos.

Inicialmente, a solução é vazia e, sucessivamente uma aresta é adicionada à solução parcial. Ao final do procedimento, a solução corresponde a um conjunto de *n* arestas, que formam uma rota que passa exatamente uma vez por cada cliente. A cada iteração, é adicionada à solução parcial a aresta de menor peso que não inviabiliza a formação de uma rota final (quais sejam, arestas que formam ciclos ou arestas que aumentam o grau de algum vértice para 3). A figura a seguir traz a aplicação da MFH (considerando já adicionados 10, 50 e 100 clientes à solução) em uma instância formada por um conjunto de 100 pontos (que representam 100 clientes).

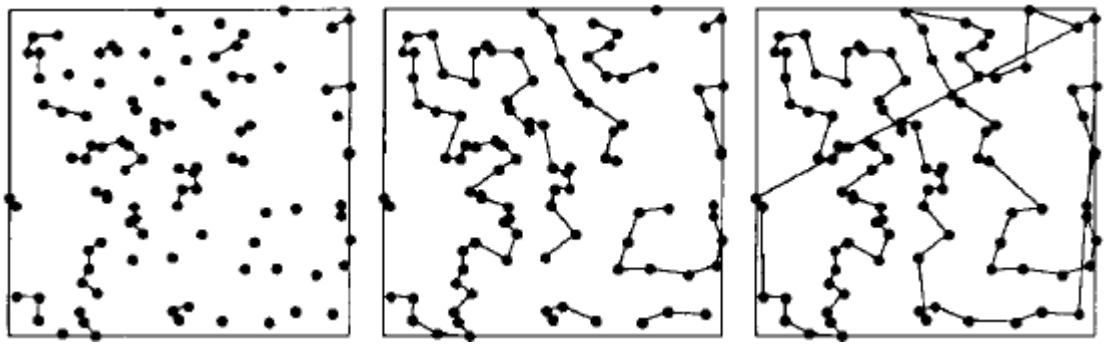

Figura 9. Rota construída pela heurística MFH. Imagem retirada de Bentley [7].

HC4: Heurística da Inserção Mais Barata (*Cheapest Insertion Heurístic* - NIH) [48]. Esta heurística se baseia em expandir uma subrota até que todos os clientes sejam visitados. Inicialmente, a subrota é composta de apenas dois clientes. A cada iteração, um novo cliente é inserido na subrota**,** até que ela contenha todos os clientes. O critério de inserção é de minimização de custo adicionado à subrota. Ou seja, encontrar os clientes *i* e *j* (vizinhos entre si e pertencentes à subrota) e *w* (não pertencente à subrota) para os quais o custo $d_{iw}+d_{jw}-d_{ij}$ seja mínimo. O primeiro cliente a ser adicionado à solução (subrota) é escolhido aleatoriamente, e o segundo, é o cliente mais próximo do primeiro.

HC5: Heurística da Inserção Mais Distante (*Farthest Insertion Heurístic* - FIH) [48]. Semelhante à HC4, esta heurística se baseia em expandir uma subrota até que todos os clientes sejam visitados, iniciando a subrota com dois clientes. A diferença está no critério de inserção: a cada iteração, o cliente *w* (não pertencente à subrota) escolhido é aquele mais distante de todos os clientes da subrota e inserido entre os dois clientes *i* e *j* (pertencentes a rota) mais próximos. O primeiro cliente a ser adicionado à solução é escolhido aleatoriamente, e o segundo, é o cliente mais distante do primeiro.

HC6: Gera uma sequência aleatória de clientes. A qualidade das soluções geradas em geral é baixa, esta heurística é utilizada para adicionar variabilidade à população inicial.

A população da primeira geração é construída com soluções geradas pelas heurísticas construtivas. Mais especificamente, as heurísticas HC1, ..., HC5 são executadas uma única vez, enquanto a heurística HC6 é executada até completar o resto da população inicial.

A ideia é utilizar diferentes abordagens construtivas que diferem quanto à estratégia de construção. Deste modo, a população inicial deve apresentar variabilidade suficiente entre suas soluções. Ainda neste sentido, com o objetivo de garantir certo nível de qualidade às soluções da população inicial, é aplicada uma busca local em cada solução produzida pelas heurísticas construtivas. A busca local é descrita na subseção 4.3.5.

O motivo para executar as heurísticas HC1, ..., HC5 uma única vez é que elas são limitadas quanto ao número de soluções que são capazes de produzir. Mais especificamente, a HC3 é determinística e capaz de produzir uma única solução. As heurísticas HC1, HC2, HC4 e HC5 são capazes de produzir, no máximo, n soluções diferentes. No entanto, a HC6 é capaz, em teoria, de produzir quaisquer das $n!$ soluções possíveis.

4.3.2 OPERADOR GENÉTICO: SELEÇÃO

Os cromossomos associados às soluções de maior qualidade, que pertencem ao Conjunto Elite, são copiados sem alteração para a próxima geração (linhas 7 e 8, algoritmo 2). A ordenação dessas soluções ocorre, considerando primeiro o objetivo primário do problema (minimização do número de viagens) e depois, o objetivo secundário (minimização da soma total das distâncias da rota).

4.3.3 OPERADOR GENÉTICO: MUTAÇÃO

A cada geração são produzidos p_m cromossomos mutantes (linha 9, algoritmo 2) através da execução de um procedimento construtivo com busca local, baseado na metaheurística GRASP. O funcionamento do algoritmo é descrito a seguir.

Constrói uma solução para o TSP adicionando, iterativamente, um novo elemento à solução parcial até que a solução esteja completa. Inicialmente, a solução parcial é composta de apenas um elemento, que é um cliente selecionado aleatoriamente de $C = \{s+1, ..., s+n\}$. A cada iteração, um novo cliente ainda não visitado (denominado v) é selecionado para ser adicionado à solução parcial. Tal cliente é um do elemento escolhido aleatoriamente da RCL que, por sua vez, é composta pelos vizinhos mais próximos de v que não foram visitados. Após adicionar o cliente v à solução parcial, este cliente é marcado como visitado e a RCL é atualizada.

O tamanho da RCL varia conforme o custo de adição dos elementos que ainda não fazem parte da solução parcial, e conforme o valor de um parâmetro α. Seja g_{min} a distância de

v para o cliente mais próximo não visitado e seja g_{max} a distância de v para o cliente mais distante não visitado. A RCL (passo 5 do procedimento 3) é composta por todos os clientes não visitados que apresentam distância de v inferior a um valor $\mu = g_{min} + \alpha\ (g_{max} - g_{min})$. A seguir é apresentado o pseudocódigo do procedimento construtivo.

Procedimento 3: construtivoSolucaoMutante()

```
1.  INÍCIO
2.    v = selecionarElementoAleatoriamente(C);
3.    X = {v};
4.    ENQUANTO (algum cliente não foi visitado) FAÇA
5.      criar a RCL;
6.      v = selecionarElementoAleatoriamente(RCL);
7.      X = X ∪ {v};
8.    FIM-ENQUANTO
9.  FIM
10. RETORNA X;
```

Após produzir um cromossomo mutante, aplica-se um procedimento de busca local descrito na seção 4.2.5.

4.3.4 OPERADOR GENÉTICO: CRUZAMENTO

O operador genético de Cruzamento é aplicado para produzir cromossomos filho a partir da combinação de dois cromossomos, considerando cromossomos do Conjunto Elite ($|P_E| = p_e$) e do Conjunto Não-Elite ($|P_{NE}| = p - p_e$), selecionados aleatoriamente (e com reposição). A cada geração são realizados $p - p_e - p_m$ cruzamentos para preencher parte da população da próxima geração com cromossomos filho. Essencialmente, um cruzamento é um procedimento que recebe em sua estrutura de entrada dois vetores (cromossomo elite X_1 e cromossomo não-elite X_2) e retorna um vetor, denominado cromossomo filho X_{filho}. Esse vetor deve conter, majoritariamente, características do vetor elite. Cada vetor X_1, X_2 e X_{filho} tem tamanho n e é composto por uma sequência de clientes. Sendo assim, os cruzamentos aqui apresentados poderiam ser aplicados ao TSP.

Após a aplicação dos $p - p_e - p_m$ vetores filhos a partir do cruzamento, inicia-se o procedimento de busca local, descrito na subseção 4.3.5.

No caso do algoritmo proposto (linhas 15 até 17, algoritmo 2) nesta dissertação, foram desenvolvidos três procedimentos de cruzamento, descritos a seguir:

Cruzamento1: Inicialmente, a solução filho contém apenas um cliente, selecionado aleatoriamente de C. A cada iteração, um novo cliente é selecionado para ser adicionado à solução filho até que ela contenha todos os n clientes. O cliente adicionado a cada iteração é o vizinho à direita do último cliente adicionado, de acordo com a ordenação de um vetor X. O

vetor X, utilizado a cada iteração, é o vetor elite X_1 com probabilidade p_c (linhas 7 e 8, Procedimento 4), ou o vetor não-elite X_2 com probabilidade 1- p_c (linhas 9 e 10, Procedimento 4). Uma função, descrita a seguir, é utilizada a cada iteração para definir o cliente a ser adicionado.

selecionarProximoElemento – recebe como parâmetro de entrada v, o último cliente adicionado à solução filho e um vetor X. Retorna o próximo cliente não visitado à direita de v. Caso o próximo cliente já tenha sido visitado, retorna um cliente não visitado que possui menor distância de v, dentre todos os clientes não visitados. A figura a seguir ilustra um exemplo para diferentes parâmetros de entrada v. Neste caso, supondo $v = c_3$, a função retornaria c_7; supondo $v = c_6$, a função retornaria c_4.

X	1	5	3	7	6	4	2
selecionarProximoElemento	5	3	7	6	4	2	1

Figura 10. Exemplo de aplicação da função selecionarProximoElemento.

Procedimento 4: Cruzamento1(X_1, X_2, ρ_c)

1. **INÍCIO**
2. X_{filho} = {};
3. v = selecionarElementoAleatoriamente(X_1);
4. X_{filho} = {v};
5. **ENQUANTO** (algum cliente não foi adicionado) **FAÇA**
6. rng = gerarNumeroAleatorio(0,1);
7. **SE** (rng < ρ_c) **ENTÃO**
8. v = selecionarProximoElemento(v, X_1);
9. **SENÃO**
10. v = selecionarProximoElemento(v, X_2);
11. **FIM-SE**
12. X_{filho} = X_{filho} ∪ {v};
13. **FIM-ENQUANTO**
14. **RETORNA** X_{filho};

Cruzamento2: Este procedimento também constrói a solução filho de modo iterativo. A cada iteração, uma entrada do vetor solução filho é preenchida com um cliente que pode vir da solução elite, com probabilidade ρ_c ou não-elite com probabilidade 1 - ρ_c. A figura a seguir apresenta este procedimento, para n = 7 clientes, supondo ρ_c = 0,7.

		Solução Elite (vetor X_1)						
		2	**5**	7	**3**	1	4	**6**
		Solução Não-Elite (vetor X_2)						
		4	3	**2**	7	**6**	**5**	1
		Números Aleatórios no Intervalo [0,1]						
		0,62	0,15	**0,89**	0,23	**0,91**	**0,73**	0,44
iteração	vetor	Solução Filho (vetor X_{filho})						
1	X_1	2	-	-	-	-	-	-
2	X_1	2	5	-	-	-	-	-
3	X_2	2	5	2	-	-	-	-
4	X_1	2	5	2	3	-	-	-
5	X_2	2	5	2	3	6	-	-
6	X_2	2	5	2	3	6	5	-
7	X_1	2	5	2	3	6	5	6

Figura 11. Construção de um cromossomo filho produzido pelo Cruzamento2.

A consequência disto é que, ao final das *n* iterações, a solução filho contém alguns clientes visitados várias vezes e alguns clientes que não foram visitados. Para resolver este problema, aplica-se um procedimento iterativo de correção, trocando clientes que aparecem na solução mais de uma vez por clientes que não estão na solução. As trocas são feitas visando à minimização do custo, testando a cada iteração todas as possibilidades de trocas de clientes e escolhendo a troca de menor custo.

A figura 12 a seguir ilustra o funcionamento da correção para a solução construída na figura 11.

	Solução Filho (vetor X_{filho})						
iteração	2	5	2	3	6	5	6
1	2	5	**1**	3	6	5	6
2	2	5	1	3	6	5	**7**
3	2	5	1	3	6	**4**	7

Figura 12. Correção do cromossomo produzido pelo Cruzamento2.

De acordo com a figura 12, após a construção da solução verifica-se que os clientes 1, 7 e 4 não foram visitados, enquanto os clientes 2, 5 e 6 foram visitados mais de uma vez. A etapa de correção se baseia em substituir os clientes 2, 5 e 6 pelos clientes 1, 7 e 4. Na iteração 1, dentre os clientes não visitados (1, 7 e 4), o cliente 1, na posição 3, apresenta menor custo de troca.

Cruzamento3: O terceiro procedimento de cruzamento é inspirado nos cruzamentos *one-point-crossover* e *two-point-crossover* [31]. Neste caso, seleciona-se um valor inteiro *y*

aleatoriamente entre 1 e $\lfloor n * \rho_c \rfloor$. Uma sequência de clientes do cromossomo não-elite é copiada para o cromossomo filho nas mesmas posições. Esta sequência começa na posição *y* e termina na posição $y + \lceil n * (1 - \rho_c) \rceil$.

O restante do cromossomo filho é preenchido, segundo a ordenação de clientes da solução elite que ainda não foram adicionados à solução filho.

A figura 13 a seguir ilustra o funcionamento deste cruzamento, com *n* = 7 clientes e ρ_c = 0,9. O valor inteiro *y* selecionado aleatoriamente entre 1 e $\lfloor 7 * 0{,}9 \rfloor$ foi 3, ou seja, a sequência de clientes copiada do cromossomo elite começa na terceira posição do vetor e termina na posição $3 + \lceil 7 * (1 - 0{,}9) \rceil = 4$.

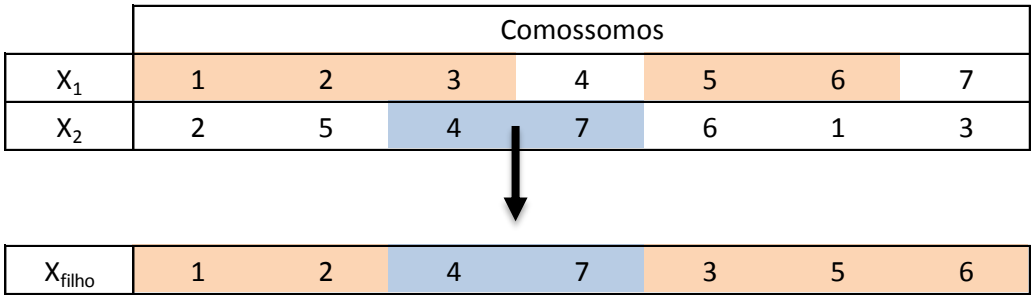

Figura 13. Construção de um cromossomo produzido pelo Cruzamento3.

A sequência escolhida contém os clientes 4 e 7, que estão nas posições 3 e 4 e são copiados do cromossomo não-elite para o cromossomo filho. As demais posições são preenchidas com os clientes não selecionados, segundo a ordenação do cromossomo elite.

4.3.5 REFINAMENTO DO CROMOSSOMO

Durante a aplicação do operador genético de cruzamento, após produzir qualquer cromossomo filho, aplica-se neste cromossomo um procedimento de busca local (linhas 19 e 21, algoritmo 2). A busca local também é aplicada nos cromossomos da população inicial (linha 4, algoritmo 2) e nos cromossomos mutantes (linha 9, algoritmo 2).

Tal procedimento é aplicado antes da escolha dos hotéis e do cálculo da função objetivo, sem levar em consideração o limite de tempo *TL*. Ou seja, trata-se de uma busca local semelhante àquelas do TSP clássico. A busca local é aplicada aos cromossomos da população inicial e àqueles produzidos pelos operadores de mutação ou cruzamento. Assim, o refinamento das soluções é feito por meio da busca local aplicada à sequência de clientes, buscando minimizar a soma total das distâncias entre eles.

Conforme definido, o tempo total gasto em uma viagem não pode exceder o limite para jornada de trabalho *TL*, sendo o tempo total de uma viagem dado pela soma dos tempos de deslocamento do vendedor somado ao tempo de atendimento dos clientes da viagem. Os tempos de atendimento dos clientes é dado no problema, sendo necessário atender a cada um

deles uma vez. Sendo assim, existe um custo de tempo fixo gasto ao longo das viagens, dado pelo atendimento de todos os clientes. Já o tempo gasto no deslocamento do vendedor entre clientes e hotéis ao longo da rota pode ser reduzido.

Uma estratégia eficaz para reduzir o tempo de deslocamento entre clientes é a aplicação da busca local na ordenação de clientes. A busca local aqui utilizada, tem como objetivo, diminuir o tamanho total da rota de clientes, permitindo que, por sua vez, durante o processo de partição da rota em viagens**,** seja possível produzir uma solução com quantidade menor de viagens sem violar o limite *TL*. Portanto, a aplicação da busca local é feita sobre a sequência ordenada de clientes (e não na solução do TSPHS), como se estivesse sendo aplicada a uma solução do TSP. Deste modo, as estruturas de vizinhança estão associadas à alteração da ordem de visita de clientes e nunca de hotéis.

Seja *X* um cromossomo, x_h é a *h*-ésima entrada deste cromossomo e contém o índice do *h*-ésimo cliente visitado na rota. A busca local tem por objetivo minimizar a função *F(X)*.

$$F(X) = \sum_{h=0}^{n-2} dist(x_h, x_{h+1}) \tag{4.2}$$

A vantagem da aplicação das estruturas de vizinhança para clientes antes da aplicação do procedimento que define os hotéis é que não há uma necessidade de planejamento de construção de rotas com clientes e hotéis; antes, a solução é construída em duas etapas: primeiramente a ordenação dos clientes e depois a definição das viagens com hotéis. Esta estratégia é possível, pois, o método aqui utilizado para partição da rota, introduz os hotéis de forma ótima dada a sequência ordenada de clientes. Destarte, a ordem de visita aos clientes é a única característica passada como herança nos cruzamentos. Ou seja, os hotéis visitados nas soluções pais não são passados adiante para as soluções filho durante os cruzamentos. Este tipo de adaptação à visita de hotéis permite explorar diversas soluções, com diferentes combinações de hotéis, sem necessidade de utilizar estruturas de vizinhança para hotéis.

Para a aplicação do operador de cruzamento, utiliza-se um parâmetro que indica a probabilidade de um cromossomo filho ser utilizado na busca local, sendo este parâmetro menor ou igual a 50%. A justificativa para isto é que a busca local é um dos procedimentos mais custosos do algoritmo. Além disso, a menor rota possível, considerando apenas os clientes e o hotel inicial (sem o limite de tempo *TL*), não necessariamente corresponde à ordenação de clientes da solução ótima do TSPHS. Em geral, realizar um movimento que reduz a distância entre clientes no cromossomo, implica melhorar a qualidade da solução que será produzida para o TSPHS, mas pode haver exceções.

O procedimento utilizado foi baseado no RVND (*Random Variable Neighborhood Descent*), e consiste, em linhas gerais, em utilizar diversas estruturas de vizinhança que são exploradas a partir de uma solução. Uma determinada estrutura de vizinhança é explorada iterativamente até que não haja mais nenhum movimento que aprimore a solução, então, a próxima estrutura de vizinhança é aplicada. Neste caso, quando ocorre alguma melhoria na solução, a busca retorna à primeira vizinhança [28, 29]. No RVND, a ordem de aplicação das estruturas de vizinhança é aleatória.

Todas estruturas de vizinhança aqui utilizadas são baseadas nos movimentos do tipo *k*-opt, aplicados a clientes. Este tipo de movimento é aplicado sobre uma rota de clientes, onde cada cliente é representado por um vértice e cada ligação entre clientes é representada por uma aresta. Um movimento *k*-opt é equivalente a remover *k* arestas da rota de clientes e adicionar *k* novas arestas a fim de criar nova rota. A implementação mais simples do 2-opt, 3-opt e 4-opt, que leva em consideração todos os movimentos possíveis a serem avaliados, tem complexidade de $O(n^2)$, $O(n^3)$ e $O(n^4)$ respectivamente. Além disso, após selecionar as arestas a serem removidas, existem $(k-1)!\,2^{k-1}$ maneiras diferentes de reconectar as arestas [8]. Quanto mais estruturas de vizinhança forem utilizadas na busca local, maiores as chances de se encontrar um ótimo local de alta qualidade, sendo esta qualidade referente a função objetivo do TSP. Para evitar buscas muito demoradas, o espaço de busca foi limitado para arestas que ligam clientes a até *r* clientes vizinhos mais próximos. Esta modificação reduz a complexidade para $O(nr)$ [41].

A restrição de considerar, durante a busca local, apenas os vizinhos mais próximos, pode impactar na qualidade da solução final. No entanto, há uma redução expressiva quanto ao tempo de processamento sem uma grande perda de qualidade dependendo da escolha do valor de *r*. Segundo Nilsson [41], para $r = 5$, geralmente há uma redução pequena quanto à qualidade final da rota em detrimento a um grande aumento de velocidade na busca. Por este motivo utilizou-se $r = 5$ neste trabalho. As estruturas de vizinhança são descritas a seguir.

2-opt [16] - duas arestas são removidas e reconectadas de forma diferente, de modo que a ordem de visita dos clientes internos às arestas é invertida. Neste caso, existe apenas uma forma de reconectar as arestas além da original. Na figura 14 a seguir são apresentados os dois possíveis casos de alocação de arestas para o movimento 2-opt.

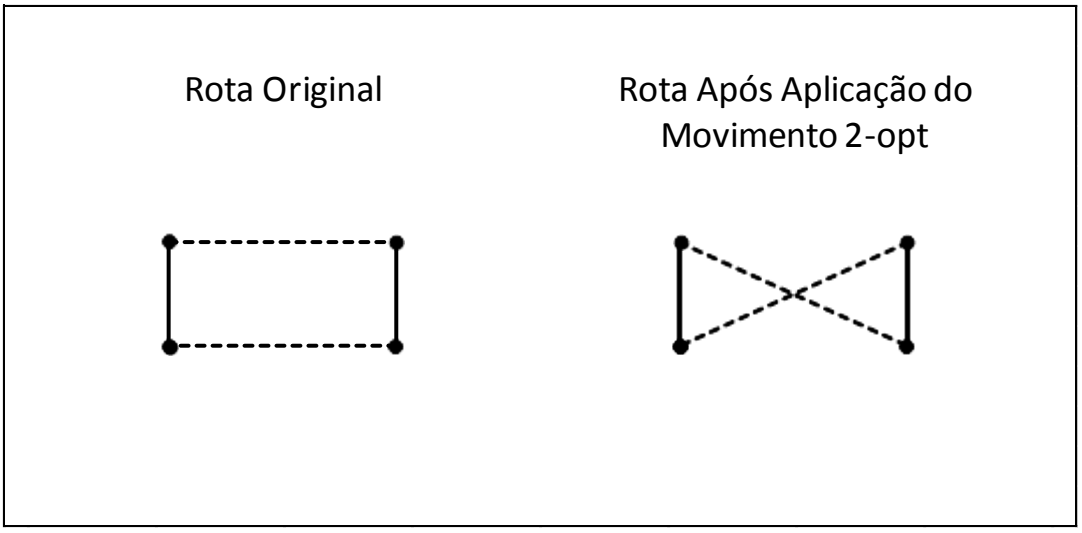

Figura 14. Exemplo de aplicação do movimento 2-opt.

Or-opt [42] - Realoca uma quantidade $c = \{1, 2, 3\}$ de clientes consecutivos.

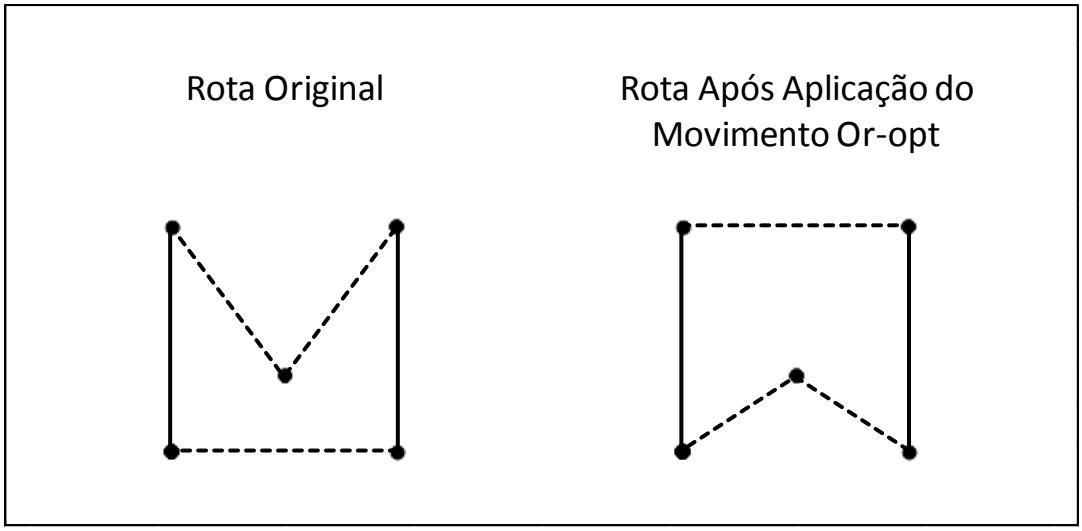

Figura 15. Exemplo de aplicação do movimento Or-opt. Neste movimento apenas um cliente foi realocado.

As linhas tracejadas nas figuras 14 e 15 representam as arestas removidas e adicionadas na rota.

Procedimento 5: RVND(X)

1. **INÍCIO**
2. Definir a Lista de Vizinhanças: $LV \leftarrow \{N_1, N_2\}$;
3. **ENQUANTO** ($LV \neq \emptyset$) **FAÇA**
4. Escolher uma vizinhança $N_k \in LV$ aleatoriamente;
5. X' = buscaLocal(N_k, X);
6. **SE** (f(X') < f(X)) **ENTÃO**
7. $X \leftarrow X'$;
8. $LV \leftarrow \{N_1, N_2\}$;
9. **SENÃO**
10. Remover N_k de LV;
11. **FIM-SE**
12. **FIM-ENQUANTO**
13. **RETORNA** X;

Com o objetivo de intensificar a busca por soluções de maior qualidade, foi desenvolvido um outro procedimento de busca local mais custoso para ser aplicado no lugar do RVND apenas uma vez por geração, em algum dos cromossomos filho selecionado aleatoriamente (linha 19, algoritmo 2).

Tal procedimento de busca local é baseado no VND [28, 29] no qual diversas estruturas de vizinhança são exploradas a partir de uma solução. No VND, a ordem de aplicação das vizinhanças é definida de modo determinístico.

Procedimento 6: VND-4opt(X)

1. **INÍCIO**
2. Definir a Lista de Vizinhanças: $LV \leftarrow \{N_2, N_3, N_4\}$;
3. $k \leftarrow 2$;
4. **ENQUANTO** (k <= 4) **FAÇA**
5. X' = buscaLocal(N_k, X);
6. **SE** (f(X') < f(X)) **ENTÃO**
7. $X \leftarrow X'$;
8. $k \leftarrow 2$;
9. **SENÃO**
10. $k \leftarrow k + 1$;
11. **FIM-SE**
12. **FIM-ENQUANTO**
13. **RETORNA** X;

Deve-se atentar para o fato que, a solução obtida após a aplicação do VND corresponde a um ótimo local para cada uma das vizinhanças utilizadas. Deste modo, as chances de encontrar um ótimo local de alta qualidade (referente ao TSP) são maiores ao utilizar o VND no lugar de uma única estrutura de vizinhança [29]. No entanto, um número

grande de estruturas de vizinhança pode resultar em um custo elevado no tempo de processamento.

A definição da ordem de exploração das vizinhanças foi decidida com base no tamanho do espaço de busca de cada vizinhança em ordem crescente. Ou seja, as vizinhanças menores são exploradas primeiro. A motivação para esta estratégia é reduzir o tempo de processamento, pois as primeiras vizinhanças são exploradas mais vezes que as últimas; assim, as maiores vizinhanças (mais demoradas) serão avaliadas menos vezes.

3-opt [33] - três arestas são removidas, então são reconectadas de todas as formas possíveis e a melhor solução gerada dentre todas é selecionada. Neste caso, existem 8 formas possíveis de alocação das arestas, no entanto, apenas 4 são levadas em consideração além da alocação original. As 3 opções de alocação restantes são semelhantes a movimentos 2-opt e, portanto, desconsideradas da avaliação.

4-opt [8] - quatro arestas são removidas, então são reconectadas de todas as formas possíveis e a melhor solução gerada dentre todas é selecionada. Neste caso, existem 48 formas possíveis de alocação das arestas, no entanto, apenas 25 combinações oferecem 4 arestas novas, diferentes daquelas originais. As 23 opções de alocação restantes são semelhantes a movimentos 2-opt e 3-opt além da alocação original e, portanto, desconsideradas da avaliação.

Além de selecionar as estruturas de vizinhança, existem diversas estratégias utilizadas na literatura para realização de uma busca em vizinhança, que podem influenciar tanto na eficiência (ou velocidade) da busca quanto na eficácia. A seguir são descritas algumas estratégias de busca:

Melhor Aprimoramento (do inglês, *best improvement*) [23] – analisar todos os movimentos possíveis dentro do espaço de busca, e aplicar aquele que melhor contribui para a melhoria da solução. A vantagem desta estratégia é a garantia de que todo movimento aplicado será o melhor dentre todas as possibilidades, enquanto a desvantagem é o alto custo computacional, causado pela busca exaustiva realizada a cada iteração.

Primeiro Aprimoramento (do inglês, *first improvement*) [23] – analisar todos os movimentos possíveis até encontrar o primeiro movimento que apresenta melhoria da solução e aplicar este movimento. A vantagem desta estratégia é que, geralmente, a vizinhança não é explorada por completo, resultando em um baixo tempo de processamento da busca. A desvantagem é que, diferentemente do *best improvement*, não há nenhuma garantia quanto à qualidade do movimento aplicado, podendo existir diversas outras possibilidades de movimento que não foram exploradas.

Aprimoramento Múltiplo (do inglês, *multiple improvement*) [21] – uma outra desvantagem da estratégia *best improvement*, é que apenas um movimento (o melhor encontrado) é aplicado por iteração, enquanto todos os outros são descartados. Na estratégia *multiple improvement*, diversos movimentos podem ser aplicados a cada iteração, desde que sejam independentes entre si, ou seja, não interfiram uns nos outros. A vantagem é que, apesar do alto custo de se avaliar toda vizinhança, múltiplas melhorias encontradas ao longo da iteração são aproveitadas. Esta é uma estratégia que busca acelerar a *best improvement*.

Neste trabalho, a estratégia utilizada em todas as estruturas de vizinhança no RVND e no VND é a de Melhor Aprimoramento (*best improvement*), o que garante que todo movimento executado por iteração é o melhor dentre todas as opções possíveis para aquela vizinhança. O espaço de busca foi substancialmente reduzido pela restrição dos vizinhos mais próximos, portanto, o tempo de processamento para avaliar a vizinhança não é muito grande.

CAPÍTULO 5 – TESTES COMPUTACIONAIS

O algoritmo proposto, denotado BRKGA-LS, foi implementado em linguagem C e todos os experimentos foram realizados em um computador dotado de um processador i5-6400 (2.70GHz 16GB RAM) e em sistema operacional Windows 7. As soluções produzidas pelo BRKGA-LS foram comparadas aos melhores resultados da literatura, assim como dois algoritmos recentemente propostos que apresentam resultados de alta qualidade, a saber: HDM [36] e EA-ILS [54]. O HDM apresenta os melhores resultados da literatura quanto a qualidade das soluções [17]. As comparações estão na seção 5.3.

Os parâmetros utilizados no algoritmo (p; p_e; p_m; ρ_c; α; p_{ls}) foram definidos com base em experimentos computacionais realizados previamente para calibração de parâmetros, descritos na subseção 5.2. O limite *TP* para o tempo de processamento para cada instância foi semelhante ao do algoritmo HDM. Este tempo de processamento, como critério de parada, foi utilizado inicialmente para o algoritmo MA+TS [10]. Assim como o HDM, o BRKGA-LS foi executado uma única vez para cada instância. O EA-ILS foi executado 30 vezes para cada instância, a solução escolhida foi a melhor, ordenando primeiro pelo menor número de viagens, depois a menor tempo total de viagem e por último, o menor tempo de processamento dentre as iterações.

5.1 INSTÂNCIAS

Para testar o algoritmo proposto nesta dissertação, foram consideradas 131 instâncias da literatura (de *benchmark*), também utilizadas em outros algoritmos propostos para este problema. As instâncias estão divididas em quatro conjuntos que são brevemente descritos a seguir e podem ser obtidas em http://antor.uantwerpen.be/instances-in-the-paper-a-memetic-algorithm-for-the-travelling-salesperson-problem-with-hotel-selection/

SET1 - é constituído por 6 instâncias do VRPTW (*Vehicle Routing Problem with Time Windows*) [50] e 10 do MDVRPTW (*Multi-Depot Vehicle Routing Problem with Time Windows*) [15]. As instâncias do VRPTW (c101, c201, r101, r201, rc101 e rc201) têm 100 clientes cada, enquanto as do MDVRPTW (pr1–pr10) têm entre 48 e 288 clientes. Para adaptar estas instâncias para o TSPHS, a janela de tempo dos clientes é descartada, mas o tempo de fechamento dos hotéis é utilizado para o tempo limite *TL*. Não são conhecidas na literatura soluções ótimas para essas instâncias.

SET2 - é criado a partir das instâncias do primeiro conjunto, incluindo apenas os K = 10, 15, 30 e 40 primeiros clientes, adicionando um hotel extra no primeiro cliente. Essas

instâncias são propositalmente pequenas**,** para que seja possível comparar a eficácia das heurísticas com os resultados da formulação apresentada na subseção 2.2.1, que foi capaz de produzir 27 soluções ótimas. As instâncias triviais c201, r201 e rc201 não foram incluídas por apresentarem soluções simples que necessitam de apenas uma viagem para resolver o problema. Este conjunto contém 13 x 4 = 52 instâncias. O ótimo é conhecido para 47 destas instâncias a partir da aplicação da formulação apresentada na subseção 2.2.2; para as 5 instâncias restantes, onde o ótimo é desconhecido, utiliza-se como referência para comparação um valor obtido a partir da aplicação de um método exato (descrito na subseção 2.2.2), utilizado por Castro et al. [10].

SET3 - contém instâncias derivadas do TSP, conhecidas da literatura, contendo entre 51 e 1002 clientes e 3, 5, ou 10 hotéis extras (além do ponto de partida). Essas instâncias são criadas de modo que a solução ótima é conhecida, utilizando a solução ótima do TSP para definir a localização dos hotéis e valor de *TL* no TSPHS. Este conjunto contém 16 x 3 = 48 instâncias.

SET4 - é definido a partir das mesmas instâncias do terceiro conjunto, no entanto, 10 hotéis são adicionados nas localizações dos clientes 1, 6, 11, 16, 21, 25, 31, 35, 41 e 45. O valor de *TL*, é definido como o custo total da solução ótima do TSP divido por 5. Inicialmente, este conjunto era composto por 16 instâncias, no entanto, a instância lin105 foi removida. O motivo para isto é que não é possível produzir nenhuma solução viável para esta instância [10]. Existe um cliente nesta que está localizado distante demais do hotel mais próximo, sendo impossível visitá-lo em alguma viagem que respeite o limite *TL*. Como resultado disso, este conjunto possui 15 instâncias e não são conhecidas soluções ótimas para estas instâncias.

A matriz de distâncias é calculada de modo diferente para cada conjunto de instâncias, assim o tempo gasto de deslocamento entre duas localizações nas instâncias do SET1 e SET2 apresentam uma casa decimal, enquanto nas instâncias do SET3 e SET4 o tempo gasto de deslocamento apresenta sempre valores inteiros. A forma de calcular a matriz de distâncias esta explicada no mesmo site em que as instâncias são obtidas.

5.2 CALIBRAÇÃO DE PARÂMETROS

Para que o algoritmo proposto apresente bons resultados, é importante garantir uma combinação eficaz para os valores dos parâmetros que são: (p; p_e; p_m; ρ_c; α; p_{ls}).

No BRKGA-LS, a calibração dos parâmetros ocorre de duas formas diferentes: (i) por meio de testes computacionais (p; p_e; p_m; p_{ls}) para os valores dos parâmetros de entrada do algoritmo; (ii) por auto calibração durante a execução do algoritmo ao longo das gerações (ρ_c; α).

Os valores considerados neste experimento para os parâmetros são apresentados a seguir:

Tabela 1. Possibilidade de valores para os parâmetros.

Parametros	Valores
p	{50, 100, 150}
p_e	{10%p , 20%p , 25%p }
p_m	{10%p , 20%p , 30%p }
ρ_c	{60%, 70%, 80%, 90%}
α	{5%, 7,5%, 10%}
p_{ls}	{10%, 30%, 50%}

Os parâmetros populacionais referentes ao BRKGA, quais sejam: tamanho da população considerada elite (p_e), tamanho da população mutante (p_m) e probabilidade de herdar característica do cromossomo elite no cruzamento (ρ_c), foram baseados na sugestão de valores de Gonçalves e Resende [26]. Os valores associados ao tamanho da população são menores que os sugeridos por Gonçalves e Resende [26]. A justificativa para um tamanho pequeno da população é que, neste caso, gasta-se muito tempo em procedimentos como os de busca local (que não faz parte do BRKGA originalmente proposto). Ou seja, se o tamanho da população for grande demais, o algoritmo vai gastar muito tempo de processamento para produzir soluções de alta qualidade. O parâmetro p_{ls} indica a probabilidade de aplicar a busca local a um cromossomo qualquer gerado no cruzamento.

Os valores do parâmetro α são propositalmente pequenos (mais próximos de 0 do que de 1), pois no problema em questão, é interessante construir soluções com ligações entre clientes não muito distantes. Para estes valores de α, as RCL são geralmente compostas por 3 a 20 clientes mais próximos. Outra preocupação é o tempo gasto com a busca local aplicada à solução produzida pelo método construtivo do GRASP. Para valores muito altos de α, é esperado que a qualidade da solução seja baixa e que sejam necessárias muitas iterações da busca local, até que seja produzido um ótimo local de qualidade razoável. Concluindo, foram escolhidos valores entre 5 e 10% para o parâmetro.

A cada novo cromossomo mutante a ser gerado, o BRKGA-LS seleciona um valor para o parâmetro α segundo um mecanismo probabilístico inspirado em Martí et al. [37]. Neste caso, o mecanismo probabilístico é utilizado para ponderar as chances de cada valor possível para o parâmetro α. Inicialmente, todos os valores de α tem igual probabilidade de serem selecionados. A cada novo cromossomo mutante gerado, estas probabilidades são atualizadas, segundo os resultados obtidos até então. Seja $q_\alpha(i)$ o contador com o número de cromossomos produzidos com o *i*-ésimo valor do parâmetro de α que entraram no conjunto elite (este contador inicia em zero). Conforme já apresentado na tabela 1, os possíveis valores de α são 5%, 7,5% e 10%. Seja $P_\alpha(i)$ a probabilidade do *i*-ésimo valor de α ser selecionado. As equações apresentadas a seguir foram baseadas na equação proposta em de Lu et al. [36] e Martí et al. [37]:

$$P_\alpha(i) = \frac{33 + q_\alpha(i)}{99 + q_\alpha(1) + q_\alpha(2) + q_\alpha(3)} \tag{5.1}$$

De igual modo, para cada novo cruzamento, o BRKGA-LS seleciona um valor para o parâmetro ρ_c segundo um mecanismo probabilístico. Seja $q_{\rho c}(i)$ o contador com o número de cromossomos produzidos com o *i*-ésimo valor do parâmetro de ρ_c que entraram no conjunto elite, este contador inicia em zero. Os possíveis valores de ρ_c são 60%, 70%, 80% e 90%. Seja $P_{\rho c}(i)$ a probabilidade do *i*-ésimo valor de ρ_c ser selecionado. Então:

$$P_{\rho c}(i) = \frac{25 + q_{\rho c}(i)}{100 + q_{\rho c}(1) + q_{\rho c}(2) + q_{\rho c}(3) + q_{\rho c}(4)} \tag{5.2}$$

Com o intuito de encontrar uma combinação equilibrada para os valores dos parâmetros de entrada (p; p_e; p_m; p_{ls}), o BRKGA-LS foi submetido a testes com algumas instâncias para diferentes combinações de valores para os parâmetros. Neste caso, a estratégia de calibração de parâmetros utilizada é semelhante àquela que Moro [39] utilizou. Para isso, foram escolhidas instâncias de diferentes quantidades de hotéis e de clientes, apresentadas tabela a seguir.

Tabela 2. Instâncias escolhidas para calibração dos parâmetros.

Instância	Clientes	Hotéis	SET	Tempo (s)
pr03	144	6	1	48
c101	100	6	1	24
tsp225.s3	225	4	3	10
a280.s10	280	11	3	135
berlin52.s4	52	10	4	4
pr76.s4	76	10	4	8

São 4 parâmetros neste caso, cada um deles com 3 possibilidades de valores. Então, existem ao todo, $3^4 = 81$ possibilidades de combinações de valores de parâmetros.

O BRKGA-LS foi aplicado a cada uma das 6 instâncias escolhidas 10 vezes com cada combinação de parâmetros possível, sendo assim, foram realizados um total de $6x10x3^4 = 4860$ execuções do algoritmo.

A qualidade das soluções obtidas foi avaliada a partir do cálculo do *Relative Desviation Index* (RDI), proposto por Kim [32]. A vantagem desta medida é que não é necessário conhecer o valor da solução ótima. Além disso, a medida leva em consideração o custo do melhor e o pior resultado, produzido pelo algoritmo para cada instância. Assim, o valor do RDI é padronizado conforme a ordem de grandeza dos resultados obtidos no experimento. Seja S_i a média dos custos das soluções obtidas pela *i*-ésima combinação de parâmetros. Conformo já comentado, existem 81 combinações diferentes de parâmetros, assim $i = 1, ..., 81$. Seja S_B e S_W o menor (melhor) e o maior (pior) valor de S_i respectivamente. O cálculo do RDI para *i*-ésima combinação de parâmetros é apresentada a seguir:

$$RDI_i = \frac{S_B - S_i}{S_B - S_W} \tag{5.3}$$

O RDI_i pertence ao intervalo [0,1], uma vez que $S_B \leq S_i$ e $S_i \leq S_W$. Os melhores valores de RDI_i estão próximos de 0, enquanto os piores estão próximos de 1. Os 81 valores de RDI_i foram calculados para cada uma das 6 instâncias, a partir da média das 10 execuções. A média dos 6 valores de RDI_i é calculada e as 81 médias obtidas são ordenadas em ordem crescente para encontrar a melhor combinação de parâmetros. A seguir são apresentadas as combinações correspondentes aos melhores e piores valores de RDI médio.

Tabela 3. Combinações de parâmetros com 5 melhores médias de RDI.

Média RDI	p	p_e	p_m	p_{ls}
0,180	100	25%p	10%p	30%
0,240	50	25%p	10%p	50%
0,243	50	25%p	10%p	30%
0,248	150	25%p	10%p	50%
0,250	50	20%p	10%p	50%

Tabela 4. Combinações de parâmetros com 5 piores médias de RDI.

Média RDI	p	p_e	p_m	p_{ls}
0,593	100	20%p	30%p	10%
0,605	50	20%p	30%p	10%
0,616	150	20%p	30%p	10%
0,689	100	10%p	30%p	10%
0,782	150	25%p	30%p	10%

Assim, a combinação de parâmetros que resultou no melhor desempenho do algoritmo, segundo os testes realizados nesta seção foi (p = 100; p_e = 25%p; p_m = 10%p; p_{ls} = 30%), sendo este o conjunto de valores utilizados como parâmetros de entrada do BRKGA-LS em todos os experimentos realizados nesta dissertação.

5.3 RESULTADOS

Os resultados obtidos a partir do algoritmo proposto foram comparados aos resultados dos melhores algoritmos da literatura. Para isso, foram considerados dois algoritmos, que apresentaram resultados de alta qualidade frente aos demais algoritmos da literatura, propostos em trabalhos recentes: HDM (*Hybrid Dynamic Programming Memetic Algorithm*) [36] e EA-ILS (*Efficient Adaptive Iterated Local Search*) [54]. Além disso, os resultados do BRKGA-LS também foram comparados aos melhores resultados da literatura. Para cada instância, foi considerada como a melhor solução conhecida da literatura (*Best Known Solutio*n - BKS); segundo a pesquisa feita, a melhor solução encontrada dentre os seguintes algoritmos: MA+TS [10], P-LS [9], HDM [36], EA-ILS [54]. Uma revisão contendo as diferentes estratégias utilizadas por estes algoritmos (com exceção do EA-ILS que foi proposto depois) pode ser encontrada em [17]. Para as instâncias do SET2 também foram considerados os resultados de formulações dos *papers* citados, descritos nas subseções 2.2.1 e 2.2.2. A medida de comparação aqui utilizada foi o a diferença relativa do tempo total de viagem, dada pelo *gap*. Ou seja, as soluções são comparadas de acordo com o objetivo secundário. A expressão que permite calcular o valor do *gap* para comparação de uma solução *S* com a BKS da literatura é apresentada a seguir:

$$gap = \frac{T.Total(S) - T.Total(BKS)}{T.Total(BKS)} * 100 \quad (5.3)$$

O tempo total de viagem está associado ao objetivo secundário do problema, a comparação entre as soluções por meio do *gap* só é válida quando as duas soluções apresentam o mesmo número de viagens, que é o objetivo principal do problema. Para maioria das instâncias, os algoritmos aqui comparados apresentaram o mesmo número de viagens, ou seja, o valor do *gap* pode ser usado, em geral, como uma medida de comparação.

As tabelas apresentadas a seguir trazem os resultados obtidos nos experimentos comparativos.

Os resultados referentes às instâncias do SET1 estão na tabela 5. As colunas 1 e 2 indicam o nome e o número de clientes (“*n”*). As colunas 3 e 4 indicam o número de viagens (“V”), o tempo total de viagem (“T.Total”), associados à BKS. As colunas 5, 6, 7 e 8 indicam

o número de viagens, o tempo total de viagem, o tempo de processamento (em segundos) e o *gap* (comparando com a BKS) associados ao BRKGA-LS. As colunas 9, 10, 11 e 12 indicam o número de viagens, o tempo total de viagem, o tempo de processamento e o *gap* associados ao HDM. As colunas 13, 14, 15 e 16 indicam o número de viagens, o tempo total de viagem, o tempo de processamento e o *gap* associados ao EA-ILS.

Tabela 5. Resultados para as instâncias do SET1.

		BKS		BRKGA-LS				HDM				EA-ILS			
Instância	n	V	T. Total	V	T. Total	Tempo(s)	gap(%)	V	T. Total	Tempo(s)	gap(%)	V	T. Total	Tempo(s)	gap(%)
c101	100	8	9 651,1	**8**	**9 645,0**	**25,0**	-0,06%	9	9 591,1	24,0	-0,62%	8	9 651,1	0,2	0,00%
r101	100	8	1 695,5	8	1 719,0	24,0	1,39%	8	1 695,5	24,5	0,00%	8	1 709,3	0,3	0,81%
rc101	100	8	1 673,4	8	1 675,6	29,0	0,13%	8	1 673,4	29,4	0,00%	8	1 674,1	0,3	0,04%
c201	100	3	9 559,9	**3**	**9 559,7**	**16,0**	0,00%	3	9 559,9	16,3	0,00%	3	9 561,8	0,2	0,02%
r201	100	2	1 642,8	**2**	**1 637,7**	**12,0**	-0,31%	2	1 642,8	12,4	0,00%	2	1 643,7	0,3	0,05%
rc201	100	2	1 642,7	2	1 643,0	12,0	0,02%	2	1 642,7	12,6	0,00%	2	1 642,7	0,1	0,00%
pr01	48	2	1 412,2	2	1 412,2	2,0	0,00%	2	1 412,2	2,0	0,00%	2	1 412,2	0,0	0,00%
pr02	96	3	2 543,3	3	2 543,3	18,0	0,00%	3	2 548,8	18,1	0,22%	3	2 543,3	0,1	0,00%
pr03	144	4	3 404,2	**4**	**3 401,9**	**48,0**	-0,07%	4	3 404,2	48,6	0,00%	4	3 432,2	0,4	0,82%
pr04	192	5	4 215,3	**5**	**4 211,0**	**166,0**	-0,10%	5	4 215,3	166,2	0,00%	5	4 217,4	0,5	0,05%
pr05	240	5	4 948,9	**5**	**4 931,1**	**340,1**	-0,36%	5	4 948,9	340,4	0,00%	5	4 993,6	2,6	0,90%
pr06	288	7	5 960,9	7	5 961,5	337,1	0,01%	7	5 960,9	337,6	0,00%	7	5 978,0	1,9	0,29%
pr07	72	3	2 070,3	3	2 070,3	12,0	0,00%	3	2 070,3	12,5	0,00%	3	2 070,3	0,0	0,00%
pr08	144	4	3 367,7	**4**	**3 360,8**	**67,0**	-0,20%	4	3 367,7	66,9	0,00%	4	3 389,3	0,4	0,64%
pr09	216	5	4 414,9	**5**	**4 404,5**	**234,0**	-0,24%	5	4 414,9	234,7	0,00%	5	4 459,6	1,4	1,01%
pr10	288	7	5 932,0	7	5 934,1	422,0	0,04%	7	5 932,0	422,8	0,00%	7	5 956,2	7,9	0,41%

obs: As entradas em negrito indicam os casos em que o BRKGA-LS foi capaz de produzir soluções melhores que as da literatura até então

O BRKGA-LS produziu soluções melhores (em negrito) que as da literatura em 8 das 16 instâncias do SET1, são elas: c101, c201, r201, pr03, pr04, pr05, pr08, pr09, sendo esse o conjunto de instâncias no qual o BRKGA-LS obteve melhor desempenho comparativo.

As tabelas 6, 7, 8 e 9 apresentam os resultados referentes às instâncias do SET2, para K = 10, 15, 30 e 40 clientes respectivamente. Nas colunas 3 e 4 (“Exato”) estão os resultados ótimos para estas instâncias obtidos por meio da formulação (descrita na subseção 2.2.2). Atualmente, considerando a pesquisa realizada, a solução ótima é conhecida para 47 instâncias. No caso das instâncias c101(K=30), rc101(K=30), c101(K=40), r101(K=40) e rc101(K=40) a solução ótima é desconhecida, ainda assim, é possível que a solução ótima não confirmada para estas instâncias já tenha sido encontrada [36]. Por este motivo é utilizado o BKS para fins comparativos nestes 5 casos.

Tabela 6. Resultados para as instâncias do SET2 (contendo 10 clientes).

		Exato		BRKGA-LS				HDM				EA-ILS			
Instância	n	V	T. Total	V	T. Total	Tempo(s)	gap(%)	V	T. Total	Tempo(s)	gap(%)	V	T. Total	Tempo(s)	gap(%)
c101	10	1	955,1	1	955,1	1,0	0,0%	1	955,1	1,0	0,0%	1	955,1	0,0	0,0%
r101	10	2	272,8	2	272,8	1,0	0,0%	2	272,8	1,0	0,0%	2	272,8	0,0	0,0%
rc101	10	1	237,5	1	237,5	1,0	0,0%	1	237,5	1,0	0,0%	1	237,5	0,0	0,0%
pr01	10	1	426,6	1	426,6	1,0	0,0%	1	426,6	1,0	0,0%	1	426,6	0,0	0,0%
pr02	10	1	661,9	1	661,9	1,0	0,0%	1	661,9	1,0	0,0%	1	661,9	0,0	0,0%
pr03	10	1	553,3	1	553,3	1,0	0,0%	1	553,3	1,0	0,0%	1	553,3	0,0	0,0%
pr04	10	1	476,4	1	476,4	1,0	0,0%	1	476,4	1,0	0,0%	1	476,4	0,0	0,0%
pr05	10	1	528,9	1	528,9	1,0	0,0%	1	528,9	1,0	0,0%	1	528,9	0,0	0,0%
pr06	10	1	597,4	1	597,4	1,0	0,0%	1	597,4	1,0	0,0%	1	597,4	0,0	0,0%
pr07	10	1	670,2	1	670,2	1,0	0,0%	1	670,2	1,0	0,0%	1	670,2	0,0	0,0%
pr08	10	1	573,4	1	573,4	1,0	0,0%	1	573,4	1,0	0,0%	1	573,4	0,0	0,0%
pr09	10	1	645,5	1	645,5	1,0	0,0%	1	645,5	1,0	0,0%	1	645,5	0,0	0,0%
pr10	10	1	461,5	1	461,5	1,0	0,0%	1	461,5	1,0	0,0%	1	461,5	0,0	0,0%

Tabela 7. Resultados para as instâncias do SET2 (contendo 15 clientes).

		Exato		BRKGA-LS				HDM				EA-ILS			
Instância	n	V	T. Total	V	T. Total	Tempo(s)	gap(%)	V	T. Total	Tempo(s)	gap(%)	V	T. Total	Tempo(s)	gap(%)
c101	15	1	1 452,2	2	1 452,2	1,0	0,0%	2	1 452,2	1,0	0,0%	2	1 452,2	0,0	0,0%
r101	15	2	379,8	2	379,8	1,0	0,0%	2	379,8	1,0	0,0%	2	379,8	0,0	0,0%
rc101	15	1	303,2	2	303,2	1,0	0,0%	2	303,2	1,0	0,0%	2	303,2	0,0	0,0%
pr01	15	1	590,4	1	590,4	1,0	0,0%	1	590,4	1,0	0,0%	1	590,4	0,0	0,0%
pr02	15	1	745,6	1	745,6	1,0	0,0%	1	745,6	1,0	0,0%	1	745,6	0,0	0,0%
pr03	15	1	632,9	1	632,9	1,0	0,0%	1	632,9	1,0	0,0%	1	632,9	0,0	0,0%
pr04	15	1	683,4	1	683,4	1,0	0,0%	1	683,4	1,0	0,0%	1	683,4	0,0	0,0%
pr05	15	1	621,2	1	621,2	1,0	0,0%	1	621,2	1,0	0,0%	1	621,4	0,0	0,0%
pr06	15	1	685,2	1	685,2	1,0	0,0%	1	685,2	1,0	0,0%	1	685,2	0,0	0,0%
pr07	15	1	795,3	1	795,3	1,0	0,0%	1	795,3	1,0	0,0%	1	795,3	0,0	0,0%
pr08	15	1	707,2	1	707,2	1,0	0,0%	1	707,2	1,0	0,0%	1	707,2	0,0	0,0%
pr09	15	1	771,7	1	771,7	1,0	0,0%	1	771,7	1,0	0,0%	1	771,7	0,0	0,0%
pr10	15	1	611,9	1	611,9	1,0	0,0%	1	611,9	1,0	0,0%	1	611,9	0,0	0,0%

Tabela 8. Resultados para as instâncias do SET2 (contendo 30 clientes).

		Exato		BRKGA-LS				HDM				EA-ILS			
Instância	n	V	T. Total	V	T. Total	Tempo(s)	gap(%)	V	T. Total	Tempo(s)	gap(%)	V	T. Total	Tempo(s)	gap(%)
c101*	30	3	2 863,2	3	2 863,3	2,0	0,0%	3	2 863,2	1,4	0,0%	3	2 863,2	0,0	0,0%
r101	30	3	655,2	3	672,1	2,0	2,6%	3	655,2	2,1	0,0%	3	655,2	0,0	0,0%
rc101*	30	3	705,5	4	683,8	2,0	-3,1%	3	705,5	1,7	0,0%	3	705,5	0,0	0,0%
pr01	30	1	964,8	1	964,8	1,0	0,0%	1	964,8	1,0	0,0%	1	964,8	0,0	0,0%
pr02	30	2	1 078,3	2	1 078,3	1,0	0,0%	2	1 078,3	1,0	0,0%	2	1 078,3	0,0	0,0%
pr03	30	1	952,5	1	952,5	1,0	0,0%	1	952,5	1,0	0,0%	1	952,5	0,0	0,0%
pr04	30	2	1 091,6	2	1 091,6	1,0	0,0%	2	1 091,6	1,8	0,0%	2	1 091,6	0,0	0,0%
pr05	30	1	924,7	1	924,7	1,0	0,0%	1	924,7	1,0	0,0%	1	924,7	0,0	0,0%
pr06	30	2	1 063,2	2	1 063,2	1,0	0,0%	2	1 063,2	1,0	0,0%	2	1 063,2	0,0	0,0%
pr07	30	2	1 130,4	2	1 130,4	1,0	0,0%	2	1 130,4	1,0	0,0%	2	1 130,4	0,0	0,0%
pr08	30	2	1 006,2	2	1 006,2	1,0	0,0%	2	1 006,2	1,0	0,0%	2	1 006,2	0,0	0,0%
pr09	30	2	1 091,4	2	1 091,4	1,0	0,0%	2	1 091,4	1,0	0,0%	2	1 091,4	0,0	0,0%
pr10	30	1	918,9	1	918,9	1,0	0,0%	1	918,9	1,0	0,0%	1	918,9	0,0	0,0%

* a solução ótima para estas instâncias é desconhecida, portanto utiliza-se a melhor solução conhecida da literatura

No SET2 os três algoritmos produziram soluções ótimas para maioria das instâncias deste conjunto. A facilidade de produzir soluções ótimas para a maioria das instancias deste conjunto se deve ao fato delas serem muito pequenas, apresentando sempre 40 ou menos clientes e apenas um hotel além do ponto de partida.

Tabela 9. Resultados para as instâncias do SET2 (contendo 40 clientes).

		Exato		BRKGA-LS				HDM				EA-ILS			
Instância	n	V	T. Total	V	T. Total	Tempo(s)	gap(%)	V	T. Total	Tempo(s)	gap(%)	V	T. Total	Tempo(s)	gap(%)
c101*	40	4	3 866,1	4	3 866,1	2,0	0,0%	4	3 866,1	2,0	0,0%	4	3 866,1	0,0	0,0%
r101*	40	4	862,8	4	862,8	2,0	0,0%	4	862,8	1,6	0,0%	4	862,8	0,0	0,0%
rc101*	40	4	850,3	4	851,2	2,0	0,1%	4	850,3	3,3	0,0%	4	850,3	0,0	0,0%
pr01	40	2	1 160,5	2	1 160,5	1,0	0,0%	2	1 160,5	1,0	0,0%	2	1 160,5	0,0	0,0%
pr02	40	2	1 336,9	2	1 336,9	1,0	0,0%	2	1 336,9	1,6	0,0%	2	1 336,9	0,0	0,0%
pr03	40	2	1 303,4	2	1 303,4	1,0	0,0%	2	1 303,4	1,0	0,0%	2	1 303,4	0,0	0,0%
pr04	40	2	1 259,5	2	1 259,5	1,0	0,0%	2	1 259,5	1,0	0,0%	2	1 259,5	0,0	0,0%
pr05	40	2	1 200,7	2	1 200,7	1,0	0,0%	2	1 200,7	1,0	0,0%	2	1 200,7	0,0	0,0%
pr06	40	2	1 242,9	2	1 242,9	1,0	0,0%	2	1 242,9	1,0	0,0%	2	1 242,9	0,0	0,0%
pr07	40	2	1 407,0	2	1 407,0	1,0	0,0%	2	1 407,0	1,7	0,0%	2	1 407,0	0,0	0,0%
pr08	40	2	1 222,2	2	1 222,2	1,0	0,0%	2	1 222,2	1,0	0,0%	2	1 222,2	0,0	0,0%
pr09	40	2	1 284,2	2	1 284,4	1,0	0,0%	2	1 284,4	1,0	0,0%	2	1 284,4	0,0	0,0%
pr10	40	2	1 200,4	2	1 200,4	1,0	0,0%	2	1 200,4	1,0	0,0%	2	1 200,4	0,0	0,0%

* a solução ótima para estas instâncias é desconhecida, portanto utiliza-se a melhor solução conhecida da literatura

As tabelas 10, 11 e 12 apresentam os resultados referentes às instâncias do SET3, para S = 3, 5 e 10 hotéis extras adicionados respectivamente. A terceira coluna destas tabelas apresenta o custo da solução ótima para o TSP. A princípio, as soluções ótimas do SET3 para o TSPHS são conhecidas pela forma como elas foram adaptadas do TSP. No entanto, em pelo menos uma instância, berlin52(S=10), é possível produzir uma solução que apresenta quantidade menor de viagens ao custo de uma rota maior.

Tabela 10. Resultados para as instâncias do SET3 (contendo 3 hotéis extras).

			BRKGA-LS				HDM				EA-ILS			
Instância	n	TSP	V	T. Total	Tempo(s)	gap(%)	V	T. Total	Tempo(s)	gap(%)	V	T. Total	Tempo(s)	gap(%)
eil51	51	426	4	426	0,1	0,00%	4	426	3	0,00%	4	426	0	0,00%
berlin52	52	7 542	4	7 542	0,1	0,00%	4	7 542	3	0,00%	4	7 542	0	0,00%
st70	70	675	4	675	0,2	0,00%	4	675	3,1	0,00%	4	675	0	0,00%
eil76	76	538	4	538	0,3	0,00%	4	538	3,1	0,00%	4	538	0	0,00%
pr76	76	108 159	4	108 159	0,2	0,00%	4	108 159	10,2	0,00%	4	108 159	0,1	0,00%
kroa100	100	21 282	4	21 282	0,3	0,00%	4	21 282	10,3	0,00%	4	21 282	0,1	0,00%
kroc100	100	20 749	4	20 749	0,3	0,00%	4	20 749	10,2	0,00%	4	20 749	0,1	0,00%
krod100	100	21 294	4	21 294	0,4	0,00%	4	21 294	10,4	0,00%	4	21 294	0,1	0,00%
rd100	100	7 910	4	7 910	0,3	0,00%	4	7 910	10,7	0,00%	4	7 910	0,1	0,00%
eil101	101	629	4	629	0,9	0,00%	4	629	10,6	0,00%	4	629	0,1	0,00%
lin105	105	14 379	4	14 379	0,4	0,00%	4	14 379	10,9	0,00%	4	14 379	0	0,00%
ch150	150	6 528	4	6 528	1,0	0,00%	4	6 528	10,7	0,00%	4	6 528	0,1	0,00%
tsp225	225	3 916	4	3 916	6,8	0,00%	4	3 916	10,2	0,00%	4	3 916	0,4	0,00%
a280	280	2 579	5	2 613	235,1	1,32%	4	2 583	235,3	0,16%	4	2 579	3,5	0,00%
pcb442	442	50 778	5	51 295	693,1	1,02%	4	50 778	693,2	0,00%	4	50 778	4,3	0,00%
pr1002	1002	259 045	5	262 572	3 270,2	1,36%	4	259 045	3 266,8	0,00%	4	259 045	76,3	0,00%

Tabela 11. Resultados para as instâncias do SET3 (contendo 5 hotéis extras).

			BRKGA-LS				HDM				EA-ILS			
Instância	n	TSP	V	T. Total	Tempo(s)	gap(%)	V	T. Total	Tempo(s)	gap(%)	V	T. Total	Tempo(s)	gap(%)
eil51	51	426	6	426	0,1	0,00%	6	426	3,0	0,00%	6	426	0,0	0,00%
berlin52	52	7 542	6	7 542	0,1	0,00%	6	7 542	3,0	0,00%	6	7 542	0,0	0,00%
st70	70	675	6	675	0,1	0,00%	6	675	3,1	0,00%	6	675	0,0	0,00%
eil76	76	538	6	538	0,3	0,00%	6	538	3,0	0,00%	6	538	0,0	0,00%
pr76	76	108 159	6	108 159	0,2	0,00%	6	108 159	3,0	0,00%	6	108 159	0,0	0,00%
kroa100	100	21 282	6	21 282	0,3	0,00%	6	21 282	10,2	0,00%	6	21 282	0,1	0,00%
kroc100	100	20 749	6	20 749	0,3	0,00%	6	20 749	10,3	0,00%	6	20 749	0,1	0,00%
krod100	100	21 294	6	21 294	0,5	0,00%	6	21 294	10,3	0,00%	6	21 294	0,1	0,00%
rd100	100	7 910	6	7 910	0,3	0,00%	6	7 910	10,3	0,00%	6	7 910	0,1	0,00%
eil101	101	629	6	629	0,9	0,00%	6	629	10,2	0,00%	6	629	0,1	0,00%
lin105	105	14 379	6	14 379	0,3	0,00%	6	14 379	10,0	0,00%	6	14 379	0,1	0,00%
ch150	150	6 528	6	6 528	1,4	0,00%	6	6 528	10,4	0,00%	6	6 528	0,1	0,00%
tsp225	225	3 916	6	3 916	39,1	0,00%	6	3 916	89,8	0,00%	6	3 916	0,3	0,00%
a280	280	2 579	6	2 579	50,4	0,00%	7	2 639	199,1	2,33%	6	2 582	1,1	0,12%
pcb442	442	50 778	7	51 664	490,4	1,74%	6	50 778	490,9	0,00%	6	50 778	3,0	0,00%
pr1002	1002	259 045	7	260 652	3 987,9	0,62%	6	259 045	3 987,5	0,00%	6	259 045	49,6	0,00%

Tabela 12. Resultados para as instâncias do SET3 (contendo 10 hotéis extras).

			BRKGA-LS				HDM				EA-ILS			
Instância	n	TSP	V	T. Total	Tempo(s)	gap(%)	V	T. Total	Tempo(s)	gap(%)	V	T. Total	Tempo(s)	gap(%)
eil51	51	426	10	426	0,1	0,00%	10	426	3,0	0,00%	10	426	0,0	0,00%
berlin52	52	7 542	8	7 967	5,0	5,64%	8	7 864	4,9	4,27%	8	7 870	0,1	4,35%
st70	70	675	10	675	0,2	0,00%	10	675	3,1	0,00%	10	675	0,0	0,00%
eil76	76	538	11	538	0,3	0,00%	11	538	3,0	0,00%	11	538	0,0	0,00%
pr76	76	108 159	11	108 159	0,2	0,00%	11	108 159	10,1	0,00%	11	108 159	0,0	0,00%
kroa100	100	21 282	11	21 282	0,3	0,00%	11	21 282	10,1	0,00%	11	21 282	0,1	0,00%
kroc100	100	20 749	11	20 749	0,5	0,00%	11	20 749	10,1	0,00%	11	20 749	0,1	0,00%
krod100	100	21 294	11	21 294	0,5	0,00%	11	21 294	10,1	0,00%	11	21 294	0,1	0,00%
rd100	100	7 910	10	7 910	0,3	0,00%	10	7 910	10,1	0,00%	10	7 910	0,1	0,00%
eil101	101	629	11	629	0,7	0,00%	11	629	10,1	0,00%	11	629	0,1	0,00%
lin105	105	14 379	10	14 379	0,3	0,00%	10	14 379	10,1	0,00%	10	14 379	0,1	0,00%
ch150	150	6 528	11	6 528	1,3	0,00%	11	6 528	10,7	0,00%	11	6 528	0,2	0,00%
tsp225	225	3 916	11	3 916	6,1	0,00%	11	3 916	84,9	0,00%	11	3 916	0,5	0,00%
a280	280	2 579	11	2 579	51,6	0,00%	11	2 585	135,2	0,23%	11	2 585	1,2	0,23%
pcb442	442	50 778	12	51 481	526,2	1,38%	11	50 778	526,9	0,00%	11	51 215	13,3	0,86%
pr1002	1002	259 045	12	263 489	3 321,0	1,72%	11	259 045	3 325,8	0,00%	11	259 045	34,7	0,00%

Os resultados encontrados no SET3 demonstram que o BRKGA-LS foi capaz de produzir soluções ótimas para maioria das instâncias em tempo de processamento aceitável. Para duas das instâncias maiores (em número de clientes), pcb442 e pr1002, a solução

produzida apresentou um *gap* inferior a 3%. Para todas as outras instâncias, o BRKGA- LS foi capaz de produzir soluções ótimas, com exceção apenas de berlin52(S=10).

O último conjunto é o SET4, que é considerado um conjunto com instâncias difíceis de resolver. Neste caso, o BRKGA-LS foi capaz de produzir soluções melhores que as da literatura para quatro instâncias, são elas: berlin52, kroa100, ch150 e pcb442. No entanto, o BRKGA-LS não é capaz de produzir soluções semelhantes as BKS em diversos casos.

Tabela 13. Resultados para as instâncias do SET4.

		BKS		BRKGA-LS				HDM				EA-ILS			
Instância	n	V	T. Total	V	T. Total	Tempo(s)	gap(%)	V	T. Total	Tempo(s)	gap(%)	V	T. Total	Tempo(s)	gap(%)
eil51	51	6	429	6	429	4,0	0,00%	6	433	4,2	0,93%	6	431	0,0	0,47%
berlin52	52	7	8 642	**7**	**8 619**	**4,0**	-0,27%	7	8 642	4,1	0,00%	7	8 642	0,0	0,00%
st70	70	6	718	6	726	10,0	1,11%	6	718	10,2	0,00%	6	727	0,1	1,25%
eil76	76	6	539	6	540	12,0	0,19%	6	539	12,6	0,00%	6	549	0,1	1,86%
pr76	76	6	118 061	7	116 853	8,0	-1,02%	6	118 318	8,0	0,22%	6	118 983	0,2	0,78%
kroa100	100	6	22 205	**6**	**22 074**	**20,0**	-0,59%	6	22 205	20,3	0,00%	6	22 205	0,1	0,00%
kroc100	100	6	20 933	6	20 933	12,0	0,00%	6	20 933	12,2	0,00%	6	20 933	0,2	0,00%
krod100	100	6	21 464	6	21 507	17,0	0,20%	6	21 548	17,0	0,39%	6	21 785	0,2	1,50%
rd100	100	6	8 244	6	8 258	23,0	0,17%	6	8 352	23,7	1,31%	6	8 488	0,3	2,96%
eil101	101	6	634	6	634	23,0	0,00%	6	634	23,0	0,00%	6	634	0,1	0,00%
ch150	150	6	6 578	**6**	**6 562**	**56,0**	-0,24%	6	6 578	56,1	0,00%	6	6 619	0,3	0,62%
tsp225	225	6	4 502	7	4 710	120,1	4,62%	6	4 508	120,3	0,13%	6	4 527	5,1	0,56%
a280	280	6	2 645	6	2 660	168,1	0,57%	6	2 645	168,6	0,00%	6	2 673	4,3	1,06%
pcb442	442	6	54 137	**6**	**53 927**	**890,3**	-0,39%	6	54 137	890,8	0,00%	6	54 866	16,1	1,35%
pr1002	1002	6	291 237	7	302 425	3 530,2	3,84%	7	289 337	3 529,5	-0,65%	6	291 237	478,7	0,00%

obs: As entradas em negrito indicam os casos em que o BRKGA-LS foi capaz de produzir soluções melhores que as da literatura até então

Deve-se notar que, o algoritmo foi capaz de produzir uma solução melhor que aquelas da literatura para a instância pcb442. No entanto, o algoritmo não foi capaz de produzir o ótimo para nenhuma das variantes desta mesma instância no SET3. Isto evidencia como a posição dos hotéis, assim com o tempo limite *TL*, tem influência no desempenho de um algoritmo.

Para todas as comparações a seguir, os valores das BKS foram atualizados considerando os resultados do BRKGA-LS. A figura a seguir traz, por conjunto de instâncias, o percentual de instâncias em que cada algoritmo produziu o valor da BKS (ou solução ótima quando é conhecida).

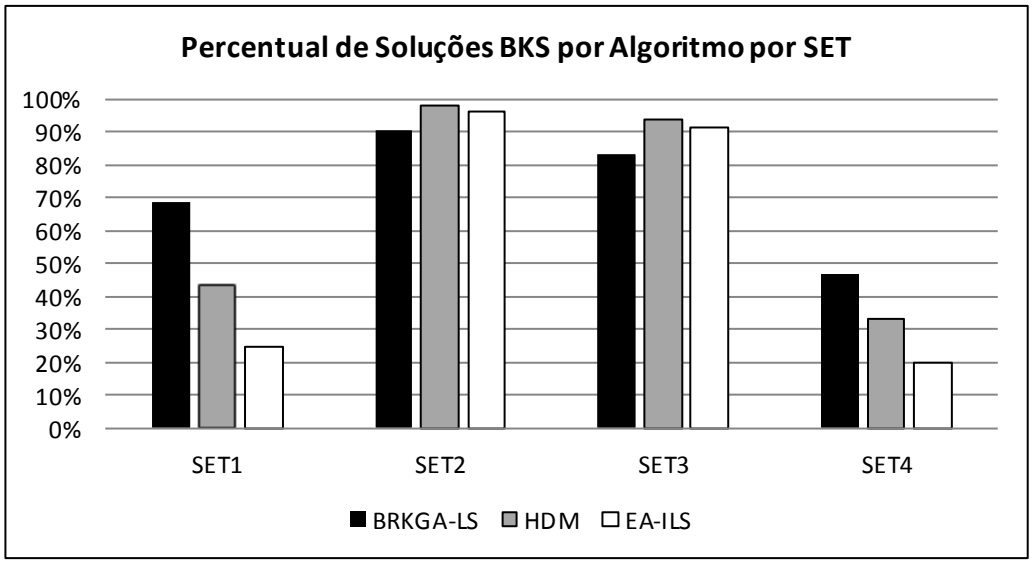

Figura 16. Resultados comparativos separados por conjunto de instâncias.

No SET3, o BRKGA-LS foi capaz de produzir o ótimo para a maioria das instâncias, com exceção das maiores. No SET4, composto pelas instâncias mais desafiadoras, o algoritmo foi capaz de produzir quatro soluções melhores que as da literatura. Os resultados agrupados por conjunto de instâncias são apresentados a seguir na Tabela 14. A primeira linha apresenta a fração do número de vezes em que cada algoritmo produziu a solução ótima, para o SET2 e SET3, onde as soluções ótimas são conhecidas. A segunda linha apresenta a fração do número de vezes em que cada algoritmo produziu a BKS, para o SET1 e SET4. Na 3ª e 4ª linhas estão, respectivamente, os *gap*s e tempos de processamento (em segundos) médios.

Tabela 14. Resumo dos resultados separados por conjunto de instâncias.

	SET1			SET2			SET3			SET4		
	BRKGA-LS	HDM	EA-ILS	BRKGA-LS	HDM	EA-ILS	BRKGA-LS	HDM	EA-ILS	BRKGA-LS	HDM	EA-ILS
Solução Ótima	-	-	-	45 / 47	46 / 47	45 / 47	40 / 48	45 / 48	44 / 48	-	-	-
BKS	11 / 16	7 / 16	4 / 16	2 / 5	5 / 5	5 / 5	-	-	-	7 / 15	5 / 15	3 / 15
gap médio	0,10%	0,06%	0,40%	-0,01%	0,00%	0,00%	0,22%	0,06%	0,03%	0,65%	0,26%	0,93%
Tempo médio (s)	110,27	110,56	1,04	1,12	1,16	0,00	264,52	277,58	3,97	326,51	326,71	33,72

O BRKGA-LS produziu soluções ótimas para a maioria das soluções dos conjuntos SET2 e SET3. As soluções ótimas do SET3 seguem a mesma ordenação de clientes que as soluções ótimas de suas respectivas versões do TSP clássico. Por conta disso, é esperado que a estratégia de diminuir o tamanho das rotas por meio de buscas locais em clientes seja efetiva e quase suficiente para se chegar à solução ótima do TSPHS. Esta característica é particular das instâncias deste conjunto por causa do modo como foram construídas.

O SET1 e o SET4 não têm esta facilidade, por apresentarem quantidade grande de clientes em algumas instâncias e posicionamento arbitrário de hotéis. Deste modo, a ordenação de clientes que minimiza a distância total da rota, não necessariamente corresponde à ordenação que minimiza o número de viagens. No caso destes conjuntos, o BRKGA-LS obteve o melhor desempenho comparativo.

A ordenação de clientes e hotéis deve ser simultaneamente definida. Neste caso, o BRKGA-LS foi capaz de produzir diversas soluções melhores que as da literatura, pois não utiliza os hotéis na representação das soluções; antes, introduz os hotéis de forma ótima dada a sequência ordenada de clientes no momento do cálculo da qualidade da solução.

Na prática, isto significa que uma solução, que surge a partir de um cromossomo filho do cruzamento, pode definir visita a hotéis diferentes daqueles presentes nas soluções dos cromossomos pais do cruzamento. Este tipo de adaptação facilita encontrar soluções boas para conjuntos difíceis como o SET1 e SET4. O *gap* médio das soluções produzidas pelo BRKGA-LS (0,16%), considerando todas as 131 instâncias, é próximo do HDM (0,06%) e EA-ILS (0,17%). Além disso, o *gap* médio do BRKGA-LS em cada um dos conjuntos também é baixo. Isto ocorre porque o algoritmo é capaz de produzir soluções de alta qualidade constantemente com *gap* próximo de 0% quando comparadas às BKS.

Os três algoritmos produziram a BKS (ou solução ótima, para os casos em que é conhecida) em 77 a 82% das instâncias aproximadamente. Sendo assim, terceiro quartil dos *gaps* dos três algoritmos aqui comparados, considerando os 4 conjuntos de instâncias, é zero (logo, o *gap* mediano também é zero), ou seja, os três algoritmos aqui comparados são capazes de produzir soluções de alta qualidade sistematicamente.

A comparação entre as soluções produzidas pelo EA-ILS e os outros algoritmos deve ser feita com ressalvas. O EA-ILS foi executado 30 vezes, apresentando a melhor das soluções produzidas, o que é uma vantagem. No entanto, o tempo de execução do EA-ILS é significativamente menor que os outros dois algoritmos aqui comparados.

Por fim, é apresentada uma tabela contendo o resumo do desempenho de alguns dos melhores algoritmos encontrados na literatura, segundo a pesquisa realizada. A primeira coluna (“Algoritmo”) identifica o algoritmo, a segunda coluna (“Ano”) se refere ao ano em que o trabalho foi publicado. A terceira (“% BKS (ou ótima)”) coluna se refere ao percentual de soluções BKS (ou ótimas) encontradas pelo algoritmo. A quarta (“*gap* médio”) e quinta (“Tempo médio (s)”) colunas se referem à média dos *gaps* e tempos de processamento em segundos.

Tabela 15. Resumo dos resultados.

Algoritmo	Ano	% BKS (ou ótima)	gap médio	Tempo médio (s)
BRKGA-LS	2020	80,15%	0,16%	148,22
HDM	2019	82,44%	0,06%	153,08
EA-ILS	2018	77,10%	0,17%	5,44
P-LS	2015	67,94%	0,29%	0,98
MA+TS	2013	77,86%	0,14%	150,82

Pode-se perceber que o BRKGA-LS apresenta o segundo melhor resultado em termos de percentual de soluções BKS (ou ótimas). Além disso o *gap* médio é um dos menores.

Quanto à comparação entre os tempos de processamento dos algoritmos da literatura, deve-se sempre levar em consideração que existem diversas variáveis que podem influenciar nesta questão como, por exemplo, o modelo de processador e a quantidade de memória RAM, além das estruturas de dados utilizadas. Deve-se atentar para o fato de que estes são algoritmos estocásticos, podendo apresentar soluções diversificadas a cada nova execução em uma mesma instância. Assim sendo, deve-se dar mais importância a estatísticas descritivas gerais e por conjunto de instâncias, como o *gap* médio, tempo de processamento médio e percentual de soluções BKS (ou ótimas) encontradas. Estas informações estão apresentadas nas tabelas 14 e 15 e na figura 16.

CAPÍTULO 6 – COMENTÁRIOS FINAIS E TRABALHOS FUTUROS

Nesta dissertação foi proposto um novo algoritmo heurístico para resolver uma variante do clássico TSP, denominada TSPHS. Tal algoritmo combina metaheurísticas com procedimentos clássicos da literatura do TSP, como a busca local e as heurísticas construtivas.

Nos experimentos conduzidos, as soluções produzidas pelo novo algoritmo foram comparadas as melhores soluções da literatura. Para as 95 instâncias onde o ótimo é conhecido no SET2 e SET3, foram produzidas 85 soluções ótimas de 95 (tabela 14). Para o SET1, foi o único algoritmo da literatura a produzir a BKS em mais da metade das instâncias. Das 16 instâncias do SET1, o BRKGA-LS produziu a BKS em 11 instâncias, o HDM em 7, o EA-ILS e o MA+TS em 4 cada e o P-LS em apenas 2 instâncias. Para o SET4, o BRKGA-LS produziu a BKS em mais instâncias que o demais algoritmos da literatura. Neste caso, nenhum algoritmo foi capaz de produzir a BKS em mais da metade das instâncias. O BRKGA-LS também apresentou o segundo maior percentual de soluções BKS (ou ótimas), levando em consideração todas as instâncias, seguido do HDM (tabela 15).

Além disso, uma comparação de resultados foi feita com dois algoritmos competitivos recentemente propostos na literatura. Um deles prioriza soluções de alta qualidade, e o outro, baixo tempo de processamento. Os experimentos conduzidos demonstraram que o BRKGA- LS é competitivo, sendo capaz de produzir soluções de alta qualidade em tempo de processamento semelhante aos Algoritmos Meméticos da literatura (que também trabalham com população de soluções).

Concluindo, o BRKGA-LS apresenta vantagens com relação a outros algoritmos da literatura, encontrando diversas soluções melhores que as apresentadas na literatura para o SET1 e SET4, no entanto, apresenta dificuldades para encontrar o ótimo em algumas instâncias no SET3. O gap médio foi inferior ao do EA-ILS e superior ao do HDM.

Pode-se concluir que a abordagem proposta, principalmente a estratégia de representar a solução por um cromossomo contendo apenas informação referente aos clientes é efetiva e pode ser utilizada em novos algoritmos para o futuro. Como proposta para trabalhos futuros, pode-se avaliar melhor as causas da dificuldade em encontrar a solução ótima para certas instâncias específicas, visto que não foram produzidas soluções ótimas para nenhuma versão (com S = 3, 5, 10) de duas das maiores instâncias do SET3.

Outra possibilidade é testar novos procedimentos de busca local, mais custosos, com objetivo de produzir o ótimo para estas instâncias. Também deve-se investigar melhor as instâncias do SET2 onde o ótimo não é conhecido. O motivo para isto é que o BRKGA-LS

produziu soluções piores que o HDM e o EA-ILS para algumas destas instâncias, ou seja, mesmo o ótimo não sendo conhecido é possível melhorar os resultados.

Além disso, pode-se propor uma variante para o TSPHS que leve em consideração o custo de estadia em hotéis. O acréscimo desta informação tornaria o problema mais realista.

O algoritmo proposto foi desenvolvido a partir dos resultados do trabalho de Beltrão et al. [6]. Diversas modificações foram feitas a fim de melhorar os resultados. O algoritmo de Beltrão et al. [6] não é utilizado para comparação nesta dissertação, visto que o algoritmo aqui proposto é superior em termos de qualidade de soluções, fixado o mesmo tempo máximo de processamento.

AGRADECIMENTOS

Os autores agradecem o apoio dado pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES).

REFERÊNCIAS

[1] AHUJA, R. K.; MAGNANTI, T. L.; ORLIN, James B. *Network flows: Theory, algorithms and applications*. New Jersey: Rentice-Hall, 1993.

[2] APPLEGATE, D. L.; BIXBY, R. E.; CHVATAL, V.; COOK, W. J. *The traveling salesman problem: a computational study*. Princeton university press, 2006.

[3] BALTZ, A.; EL OUALI, M.; JÄGER, G.; SAUERLAND, V.; SRIVASTAV, A. Exact and heuristic algorithms for the travelling salesman problem with multiple time windows and hotel selection. *Journal of the Operational Research Society 66*, 4 (2015), 615-626.

[4] BEAN, J. C. Genetic algorithms and random keys for sequencing and optimization. *ORSA journal on computing 6*, 2 (1994), 154-160.

[5] BEKTAS, T. The multiple traveling salesman problem: an overview of formulations and solution procedures. *Omega 34*, 3 (2006), 209-219.

[6] BELTRÃO, A. P. V.; OCHI, L. S.; DE MOURA BRITO, J A. Heurística híbrida aplicada ao Problema do Caixeiro Viajante com Seleção de Hotéis. In *Simpósio Pesquisa Operacional da Marinha*, 2019, *Rio de Janeiro. Anais do SPOLM*, 2019.

[7] BENTLEY, J. J. Fast algorithms for geometric traveling salesman problems. *ORSA Journal on computing 4*, 4 (1992), 387-411.

[8] BLAZINSKAS, A.; MISEVICIUS, A. Combining 2-opt, 3-opt and 4-opt with k-swap-kick perturbations for the traveling salesman problem. In 17th *International Conference on Information and Software Technologies*, 27-29, 2011.

[9] CASTRO, M.; SÖRENSEN, K., VANSTEENWEGEN, P.; GOOS, P. A fast metaheuristic for the travelling salesperson problem with hotel selection. *4OR 13*, 1 (2015), 15-34.

[10] CASTRO, M.; SÖRENSEN, K., VANSTEENWEGEN, P.; GOOS, P. A memetic algorithm for the travelling salesperson problem with hotel selection. *Computers & Operations Research 40*, 7 (2013), 1716-1728.

[11] CASTRO, M.; SÖRENSEN, K., VANSTEENWEGEN, P.; GOOS, P. A simple grasp+ vnd for the travelling salesperson problem with hotel selection. Tech. rep., University of Antwerp, Faculty of Business and Economics, 2012.

[12] CASTRO, M.; SÖRENSEN, K., VANSTEENWEGEN, P.; GOOS, P. The multiple travelling salesperson problem with hotel selection. University of Antwerp, Faculty of Business and Economics, 2014.

[13] COELHO, L. C.; LAPORTE, G. A branch-and-cut algorithm for the multi-product multi-vehicle inventory-routing problem. *International Journal of Production Research 51*, 23-24 (2013), 7156-7169.

[14] CORDEAU, J. F.; GENDREAU, M.; LAPORTE, G. A tabu search heuristic for periodic and multi-depot vehicle routing problems. *Networks: An International Journal 30*, 2 (1997), 105-119.

[15] CORDEAU, J. F.; LAPORTE, G.; MERCIER, A. A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of the Operational research society 52*, 8 (2001), 928-936.

[16] CROES, G. A. A method for solving traveling-salesman problems. *Operations research 6*, 6 (1958), 791-812.

[17] DE SOUSA, M. M.; OCHI, L. S.; DE LIMA MARTINS, S. Traveling Salesperson Problem with Hotel Selection: A systematic review of the literature. In *Proceedings of the XV Brazilian Symposium on Information Systems*, 1-8, 2019.

[18] FEO, T. A.; RESENDE, M. G. C. Greedy randomized adaptive search procedures. *Journal of global optimization 6*, 2 (1995), 109-133.

[19] FESTA, P.; RESENDE, M. G. C. GRASP: basic components and enhancements. *Telecommunication Systems 46*, 3 (2011), 253-271.

[20] FIGLIOZZI, M. Vehicle routing problem for emissions minimization. Transportation Research Record, 2197, 1 (2010), 1-7.

[21] FOSIN, J.; CARIC, T.; IVANJKO, E. Vehicle routing optimization using multiple local search improvements. *automatika 55*, 2 (2014), 124-132.

[22] GENDREAU, M.; LAPORTE, G.; MUSARAGANYI, C.; TAILLARD, É. D. A tabu search heuristic for the heterogeneous fleet vehicle routing problem. *Computers & Operations Research 26*, 12 (1999), 1153-1173.

[23] GENDREAU, M.; POTVIN, J. Y, ET AL. *Handbook of metaheuristics*, vol. 2. Springer, 2010.

[24] GLOVER, F. Tabu search—part I. *ORSA Journal on computing 1*, 3 (1989), 190-206.

[25] GLOVER, F. Tabu search—part II. *ORSA Journal on computing 2*, 1 (1990), 4-32.

[26] GONÇALVES, J. F.; RESENDE, M. G. C. Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics 17*, 5 (2011), 487-525.

[27] HANSEN, P.; MLADENOVIC, N. Variable neighborhood search. In *Handbook of metaheuristics*. Springer, 2003, pp. 145-184.

[28] HANSEN, P.; MLADENOVIĆ, N.; BRIMBERG, J.; PÉREZ, J. A. M. Variable neighborhood search. In *Handbook of metaheuristics*. Springer, 2019, pp. 57–97.

[29] HANSEN, P.; MLADENOVIC, N.; PÉREZ, J. A. M. Variable neighbourhood search: methods and applications. *4OR 6*, 4 (2008), 319-360.

[30] HOLLAND, J. H. Genetic algorithms. *Scientific american* 267, 1 (1992), 66–73.

[31] KAYA, M. The effects of two new crossover operators on genetic algorithm performance. *Applied Soft Computing 11*, 1 (2011), 881-890.

[32] KIM, Y. D. Heuristics for flowshop scheduling problems minimizing mean tardiness. *Journal of the Operational Research Society 44*, 1 (1993), 19-28.

[33] LIN, S. Computer solutions of the traveling salesman problem. *Bell System Technical Journal 44*, 10 (1965), 2245-2269.

[34] LIN, S.; KERNIGHAN, B. W. An effective heuristic algorithm for the traveling-salesman problem. *Operations research 21*, 2 (1973), 498-516.

[35] LOURENÇO, H. R.; MARTIN, O. C.; STÜTZLE, T. Iterated local search. In *Handbook of metaheuristics*. Springer, 2003, pp. 320–353.

[36] LU, Y.; BENLIC, U.; WU, Q. A hybrid dynamic programming and memetic algorithm to the traveling salesman problem with hotel selection. *Computers & Operations Research 90* (2018), 193-207.

[37] MARTÍ, R.; DUARTE, A.; LAGUNA, M. Advanced scatter search for the max-cut problem. *INFORMS Journal on Computing 21*, 1 (2009), 26-38.

[38] MILLER, C. E.; TUCKER, A. W.; ZEMLIN, R. A. Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM) 7*, 4 (1960), 326–329.

[39] MORO, M. A. *Meta-heurísticas GRASP e BRKGA aplicadas ao problema da diversidade máxima*. Dissertação de mestrado, Instituto de Matemática Estatística e Computação Científica, Universidade Estadual de Campinas, Campinas, SP, Brasil, julho 2017.

[40] NAGY, G.; SALHI, S. Location-routing: Issues, models and methods. *European journal of operational research 177*, 2 (2007), 649-672.

[41] NILSSON, C. Heuristics for the traveling salesman problem. Tech. rep., Linkoping University, 2003.

[42] OR, I. *Traveling salesman-type combinatorial problems and their relation to the logistics of blood banking*. PhD thesis, Department of Industrial Engineering and Management Science, Northwestern University, Evanston, IL, United States of America, 1976.

[43] PRINS, C. A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research 31*, 12 (2004), 1985-2002.

[44] PRINS, C.; LACOMME, P.; PRODHON, C. Order-first split-second methods for vehicle routing problems: A review. *Transportation Research Part C: Emerging Technologies 40* (2014), 179-200.

[45] RADMANESH, M.; KUMAR, M.; NEMATI A.; SARIM, M. Solution of Traveling Salesman Problem with hotel selection in the framework of MILP-Tropical optimization. In *2016 American Control Conference (ACC)* (2016), IEEE, pp. 5593-5598.

[46] RESENDE, M. G. C.; RIBEIRO, C. C. GRASP: Greedy randomized adaptive search procedures. In *Search Methodologies*. Springer, 2014, pp. 287–312.

[47] ROSENKRANTZ, D. J.; STEARNS, R. E.; LEWIS, II, P. M. An analysis of several heuristics for the traveling salesman problem. *SIAM journal on computing 6*, 3 (1977), 563-581.

[48] SEMAAN, G. S.; ALFRADIQUE, G. A. A.; MONTEIRO, L. F.; PENNA, P. H. V. Um algoritmos ILS aplicado ao problema do caixeiro viajante backhauls. In *SIMPEP - Simpósio de Engenharia de Produção,* 2015*, Bauru - SP. Simpósio de Engenharia de Produção,* 2015.

[49] SEMAAN, G. S.; SUBRAMANIAN, A.; BRITO, J. A. M.; OCHI, L. S. Um algoritmo ILS aplicado ao Problema do Caixeiro Viajante com Coleta e Entrega. In *X Congresso Brasileiro de Inteligência Computacional,* 2011*, Fortaleza - CE. X Congresso Brasileiro de Inteligência Computacional,* 2011.

[50] SOLOMON, M. M. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations research 35*, 2 (1987), 254-265.

[51] SOUSA, M. M. *Heurísticas para o Problema do Caixeiro Viajante com Seleção de Hotéis*. Dissertação de mestrado, Universidade Federal de Viçosa, Viçosa, MG, Brasil, fevereiro 2015.

[52] SOUSA, M. M.; GONÇALVES, L. B. Comparação de abordagens heurísticas baseadas em algoritmo memético para o problema do caixeiro viajante com seleção de hotéis. *Proc. XLVI Simpósio Brasileiro de Pesquisa Operacional. Salvador, Brasil* (2014), 1543–1554.

[53] SOUSA, M. M.; OCHI, L. S.; COELHO, I. M.; GONÇALVES, L. B. A variable neighborhood search heuristic for the traveling salesman problem with hotel selection. In *2015 Latin American Computing Conference (CLEI)* (2015), IEEE, pp. 1–12.

[54] SOUSA, M. M.; OCHI, L. S.; DE LIMA MARTINS, S. An Efficient Heuristic to the Traveling Salesperson Problem with Hotel Selection. In *International Workshop on Hybrid Metaheuristics* (2019), Springer, pp. 31–45.

[55] SUBRAMANIAN, A. *Heuristic, exact and hybrid approaches for vehicle routing problems*. Universidade Federal Fluminense, Niterói, RJ, Brasil, Março 2012.

[56] VANSTEENWEGEN, P.; SOUFFRIAU, W.; SÖRENSEN, K. The travelling salesperson problem with hotel selection. *Journal of the Operational Research Society 63,* 2 (2012), 207-217.